Exploring Robotics with Global Specialties ARX ASURO Robot

Electronics, Robotics, and C-Programming with Microcontrollers

By Grant Plowman and Linda Nichols-Plowman

Published and Printed in the U.S. by:
Interactive Media Publishing
111 E. 1st Street
PO Box 1407
Phoenix, OR 97535
541-535-5552

To purchase additional copies of this book go to this website: www.exploringrobotics.com

Special Acknowledgement: This book includes extracts from the book More Fun with ASURO, Volume I, authors Gruber, Robin and Grewe, Jan, 1st Edition - March 2006. English translation by J.W. Richter. We want to thank these authors for permission to include this work and for their creativity and especially for the sample programs and ASURO functions.

Contents

SECTION 1 OVERVIEW

SECTION 2 PROTOTYPES WITH BREAD BOARD

SECTION 3 PCB AND ROBOT ASSEMBLY

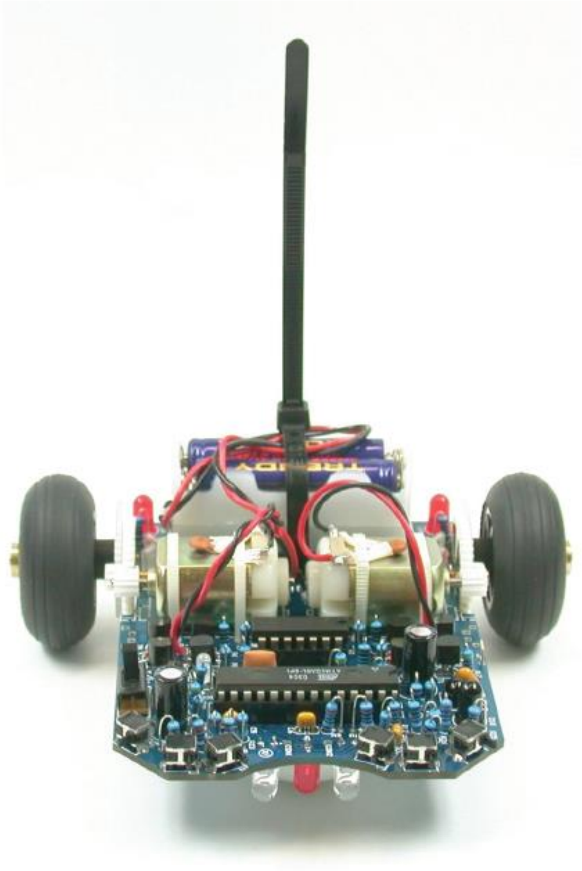
SECTION 4 PROGRAMMING

SECTION 5 COMPETITIONS

APPENDICES

SECTION 1 OVERVIEW

- *Introduction*
- *The ASURO Kit*
- *Getting to Know the ASURO*
- *The Microcontroller Brain*
- *Engineering Skills*
- *Prototyping with Breadboards*
- *Soldering PC Boards*



1. Introduction

Welcome! This text is one part of a self-paced learning package to accompany the ARX ASURO robot from Global Specialties. It is intended for ages 15 and up and is considered an intermediate level of difficulty.

Thousands of students around the world have assembled and programmed the ARX ASURO robot. It is both fun and an excellent tool for learning and applying digital electronics, C programming, and robotics concepts.

This book provides an overview and foundation to serve as a textbook. The lessons and hands-on activities provide objectives, activity instructions, video demonstrations, step-by-step assemblies, 3D CAD models, and C programs to assist in understanding the ARX ASURO parts, assemble it, and program it.

A. Introducing the ARX ASURO Robot

Let’s get started with some basic questions that most students ask in the first class.

1) What is the ARX ASURO Robot?

The Global Specialties ARX ASURO Robot (goes by either name ARX or ASURO) is a mini hobby robot that was designed and developed by the Institute for Robotics and Mechatronics in the German Aerospace Center DLE.

It has a microcontroller that is programmable with C language. Compared to other hobby robots it has few parts for its mechanical operation, which makes it FAST! Just wait till you see it run!

2) What can you do with the robot?

Besides being fun to operate wirelessly, this robot is also an excellent tool for learning about electronics, C programming, mechatronics, and robotics. The ARX ASURO has many more electronic components than most other hobby robots. Because the electronic parts need to be soldered onto the board, it is also a great way to learn soldering skills.

3) What will we accomplish and how?

We will build the robot from the ground up using the parts in the kit. We may not go as fast thru the process as you would like because we are also going to learn about engineering, electronics, mechatronics, and C programming as we go. For any skill building opportunity, it is not the destination which is important, but the journey.

4) Will it be fun?

No robot course is complete without some fun activities along the way and competitions after the robot is built. For the ARX ASURO the activities and optional competitions include line following, collision detection, maze navigation, and more.

You can also enter your ARX ASURO in competitions against other hobby robots. You will find that it is built for speed and may outperform most other similar sized robots. So yes, it will be fun!

B. About this Material

This material is broken out into the following sections:

- Section 1: Overview
- Section 2: Breadboard Prototypes
- Section 3: PCB Board and Robot Assembly
- Section 4: C Programming
- Section 5: Competitions

1) What is the Recommended Order?

We recommend going thru the material sequentially in the order presented. However, some instructors may choose to rearrange the order somewhat based upon limited class time, or skip sections based upon the level of students. If using this material with a class, refer to your syllabus.

The self-paced lessons are broken out into 5 sections or units. Some students are anxious to just build the robot and we agree that is the most fun part of this process. But we discourage it. The prototype section describes the parts used and how each of the circuits function. In it we also test each component to make sure it works.

The parts used in the breadboard prototype are used for the robot assembly. Once soldered onto the PCB board, it is not easy to remove them. That is why we highly recommend completing section 2 before moving into section 3. We recommend going thru each section in the order presented.

2) Can I Just Build the Robot?

For those who are impatient and cannot wait and just want to jump into Section 3 and begin the PCB board assembly, the purchase of two robot kits is recommended.

That way you can assemble one robot and see if you can get it to work. Then you can use the second one to go through the prototypes and learn about the components and circuits and how the robot works. You will also build the second one and gain additional practice in soldering.

We do however caution against doing this. There are some complexities to the assembly that you learn about while building the prototypes in Section 2.

3) What Are the Level and Pre Requisites?

This learning package is designed for ages 15 and up and is considered an intermediate level of difficulty.

It is helpful to have some prior exposure to PC computer applications, electronics, and C programming - or use other sources to gain that knowledge. It is also helpful to have knowledge of Algebra.

However, lack of this knowledge will not prevent you from building and programing the robot, but rather may interfere with full understanding of the processes involved.

4) What is the Learning Format?

The lessons have been designed in a self-paced format, allowing you to complete them at your own time and pace. You can work alone or with other team members to complete the lessons.

The materials are also designed to be used with project based learning. Project Based Learning is a teaching method where students gain knowledge and skills by working for an extended period of time to investigate and respond to a complex question, problem, or challenge. The project challenge is “Can we build and program a fast robot that avoids obstacles, follows lines, and operates similar to robots used in industry?” The activities also include mini-projects which provide real-world problems and challenges to solve.

STEM careers require you to think critically and solve complex problems, work collaboratively, master core academic content, communicate effectively, learn how to learn, and develop an academic mindset. These materials will help you hone those abilities.

5) What is the Estimated Time to Complete?

The estimated time for completion is included in the worksheets for each activity. Estimates are for an average student. Your time may be less or greater than these estimates depending on your background knowledge, skills, and personal motivations.

ESTIMATED TIMES:

Section 1: Getting started and building necessary skills is approx. 4 hours.
Section 2: Creating circuit prototypes is approx. 6 to 9 hours.
Section 3: Assembling the robot is approx. 6 to 10 hours.
Section 4: Completing the programming activities is approx. 10 to 15 hours.
Section 5: Participating in competitions is approx. 4 to 12 hours.

The total estimated time is 30 to 50 hours.

C. Lessons/Activities

Activity Name
Section 1: Overview
1. Introduction
2. The ARX ASURO Kit
3. Getting to Know the ARX ASURO
4. The Microcontroller Brain
5. Engineering Skills
6. Prototyping with Bread Boards
7. Soldering PC Boards
Section 2: Breadboard Prototypes
8. Power Supply Circuit
9. Status Circuit
10. Collision Detection Circuit
11. Motor Control Circuit
12. Encoder Circuit
13. Line Follower Circuit
14. Microcontroller Circuit
15. IR Communications Circuit
Section 3: PCB Board and Robot Assembly
16. Assemble Axles and Chip Sockets
17. Assemble Power Supply Circuit
18. Assemble Status Circuit
19. Assemble Collision Detection Circuit
20. Assemble Motor Control Circuit
21. Assemble Encoder Circuit
22. Assemble Line Follower Circuit
23. Assemble IR Com Link Circuit
24. Assemble Microcontroller Circuit
25. Assemble Motors and Wheels
26. Install Chips and Test Operation
27. Attach Ball and See Robot Run
Section 4: C Programming
28. Introduction to Programming
29. Our First Program
30. C Programming Essentials
31. IR Communication
32. Controlling LEDs
33. Driving – Controlling the Motors
34. Line Following

Section 5: Competitions

D. Learning Objectives

The ARX ASURO robot provides an excellent opportunity to learn and apply a variety of STEM skills which include engineering, electronics, math, physics, programming, and computer technology. Each lesson includes learning objectives. The activities are designed to direct students to meet the learning objectives. Instructors then decide if all are required, or additional objectives are added and how to assess and determine if the objectives were met.

In general, this learning package is designed to meet the following objectives, and assist students in acquiring these skills:

1) Engineering / Mechatronics skills:

- Engineering process – design, prototype, test, and finished assembly process.
- Critical thinking, troubleshooting, and prototyping
- Robotics, microcontrollers, and computer systems
- Motors, gears, torque, and gear reduction
- Technical communication and use of an Engineering notebook

2) Electronics skills:

- Electronic Symbols and reading Schematic Diagrams
- Diodes, transistors, capacitors, phototransistors, and other components
- Electrical and electronic circuits with microcontroller
- Building electronic circuits from diagrams
- Soldering and Troubleshooting solders
- Multi-meter use: reading voltage, resistance, and capacitance
- DC Power and batteries, and voltage dividers
- Printed Circuit Board and Solder-less breadboard
- Ohms law and Kirchhoff's law
- Series circuits, parallel circuits, and series-parallel circuits
- Logic Gates – AND, OR, NOR, XOR

3) Computer Science / Programming skills:

- Creating and compiling a C program and debugging programs
- Using functions and including function libraries
- Using inputs and outputs with microcontroller ports
- Variables and Memory Usage
- Number Systems – Binary, ASCII, Hexadecimal, alphanumeric
- Conditions, Loops, Pointers, Vectors, and other C language concepts

4) **Applied Physics concepts:**
   - Electric current, magnetism, electrostatics, and electromagnetic induction
   - Properties of light, reflection, light waves, and data transmitted with infrared waves
   - Force, friction, mass and weight, momentum, heat, temperature, and energy
   - Linear motion, rotational motion, torque, and acceleration

5) **Math – application of the following principles:**
   - Numbers and operations
   - Algebraic equations
   - Measurement
   - Data analysis and probability
   - Problem solving and Reasoning
   - Connections, Representation, and Communication

6) **Science – application of the following principles:**
   - The structure and properties of matter
   - The sources and properties of energy
   - Forces and motion
   - The nature of scientific inquiry
   - The scientific enterprise

E. Assessment Methods

Each instructor will choose how they will assess student's progress through the materials. The following assessment methods MAY be used. Instructors will assign points for each of these. These are listed completely in the first activity, but after that are abbreviated by their titles shown here, broken into the categories of assignments, quiz/exam, and project:

1) ASSIGNMENTS:

___/___ **Vocabulary**: Teams will be required to show evidence of examining and understanding new vocabulary words in the lesson. Teams can also generate a vocabulary list of their own that may contain more words than provided. The means of providing evidence for learning will be determined by group consensus; however evidence must reflect an application of all assigned vocabulary words by each individual. *Examples: Students divide words and make a learning activity/quiz and administer to other members, journal notations citing examples of term applied in the project; developing a group game, etc.*

___/___ **Worksheet Assignments**: Teams will be required to view videos, interact with 3D models, read the textbook, complete assignments and answer the questions in the worksheets and check off assigned items as they are completed.

___/___ **Course Research**: Teams will be responsible for locating information as defined in chosen or assigned enrichment activities or project activities. *Example: when writing a paper, quote from several sources and include at the end of the paper in a "Resources" section a list of all sources. For each source include the book or journal name or website name, title, author, pages, and date. Most papers should include at least two Internet sources, two articles from trade magazines or newspapers, and one book reference.*

___/___ **Engineering Journal Entries:** Teams will be expected to use and update their engineering journal on a daily basis. Each day's entry should have a clear log of warm-up, record of work accomplished that day, be clear, legible, and detailed and show application of lesson concepts.

___/___ **Calculations:** Teams will be required to perform calculations and show work for calculations in the work sheet and in engineering journal entries. Theory based calculations require also including the formula or theory the calculation is based upon. Tables, charts, and other methods of organizing data should be included when appropriate.

2) QUIZ/EXAM:

___/___ **Materials Covered Quiz:** The instructor may construct a quiz at any point in the process were the student will be expected to answer questions, perform and/or demonstrate their learning of the materials for this lesson or at any point in the project.

___/___ **Performance Objectives or Standards Quiz:** The instructor may construct a quiz at any point in the process were the student will be expected to answer questions, perform, and/or demonstrate meeting the performance objectives or standards for this lesson.

___/___ **Student Generated Quiz:** The instructor may use a student generated Quiz that covers the concepts and material included for this lesson.

3) PROJECT:

___/___ **Project Design:** Students will be individually assessed on their contribution to the design, testing, construction, programming, and analysis of the project. The majority of this evidence will be illustrated through documentation in the engineering notebook. Record ideas, drawings, thoughts, procedures, sketches, etc. related to the design and construction of the project, or one part of the project for this lesson. Each team member is responsible for understanding the project design and construction methods used and the teacher may ask any member to describe the project design or construction for this lesson.

___/___ **Project Demonstration or Competitions:** Students will be individually assessed on their contribution to project demonstrations or competitions. The majority of the evidence will be illustrated in the documentation submitted, performance during the demonstration or competition, and outcome of the performance or competition. Separate grading rubrics may be provided for each project demonstration or competition.

F. Materials and Resources

The following materials and resources are used with this learning package:

1) The ARX ASURO Kit

The ARX ASURO kit manufactured by Global Specialties comes with all the parts needed for assembly of the robot. The kit from Global Specialties comes with a CD that includes everything needed for programming. Other ASURO kits don't have this and may not work with this instruction.

Note: *THERE ARE NO SPARE PARTS* in the robot kit! It is very easy to drop and loose an electronic part. Use caution when handling and storing the parts. *There is a spare parts package available. We recommend it*.

2) Content - Lessons

The DVD or online content which accompanies this book contains a menu of lessons which correlate to the chapters. Each activity has instructions to follow, reading assignments, a worksheet (word doc), and links to videos and other resource materials.

While each activity will take around 1 hour to complete, it has been designed as self-study so that you may progress through the material at your own pace. If using this material as part of a class, the completed worksheet may be provided to an instructor as proof of completion of the activity.

3) **Videos**
Many of the lessons include videos. The videos provide a combination of review of concepts and demonstration of the activities. They are designed to function as if a tutor were next to the student providing individual instruction. Students can start and stop and replay the videos as needed.

4) **Software**
The CD that comes with the ARX ASURO contains some of the software to be installed. Additional software is provided as links to download or may be installed from the DVD that accompanies this learning package.

Instructions are provided for installation of software in a separate book.

5) **ARX ASURO Manual**
The manual that accompanies the ARX ASURO is included as a PDF document on the CD that comes with the robot. We have also linked it into the menu. It has been translated from German to English. With any translation there are often differences in language usage, especially with technical topics. We will refer to the technical specifications in the ARX ASURO Manual but will not follow the order shown in the manual for assembly.

6) **Drawings and Diagrams**
3D drawings of the ARX ASURO are provided to make it easy to visualize the prototypes and assembly of the robot. Also, the large schematic diagram in the back of the ARX ASURO manual was broken apart into smaller more manageable systems diagrams. There are additional drawings and diagrams in the manual which we will reference.

7) **Example Programs**
Programs and documentation of programs are provided to assist students in getting started with the programming activities. These are either downloaded with the lessons or may be copied to the computer as part of the software installation process.

8) **Internet Resources**
Additional links are provided to other resources on the Internet. These often provide background information and further technical details of a topic.

G. Tools Required

With any engineering project, it is best to assemble the tools and resources prior to beginning the building process. The following list of tools is required.

1) Tools for Completing Assignments and Programming (Section 4):

- Computer: a laptop or personal computer with Internet access and Windows operating system. (While the ARX software supports Windows or Linux, this training only supports Windows).
- C language development software (on CD that comes with the ARX ASURO)
- Other software that comes with the ARX ASURO CD or the activities: Flash, Make, and Hyperterminal programs
- Sketchup drawing software (free download) for viewing 3D diagrams

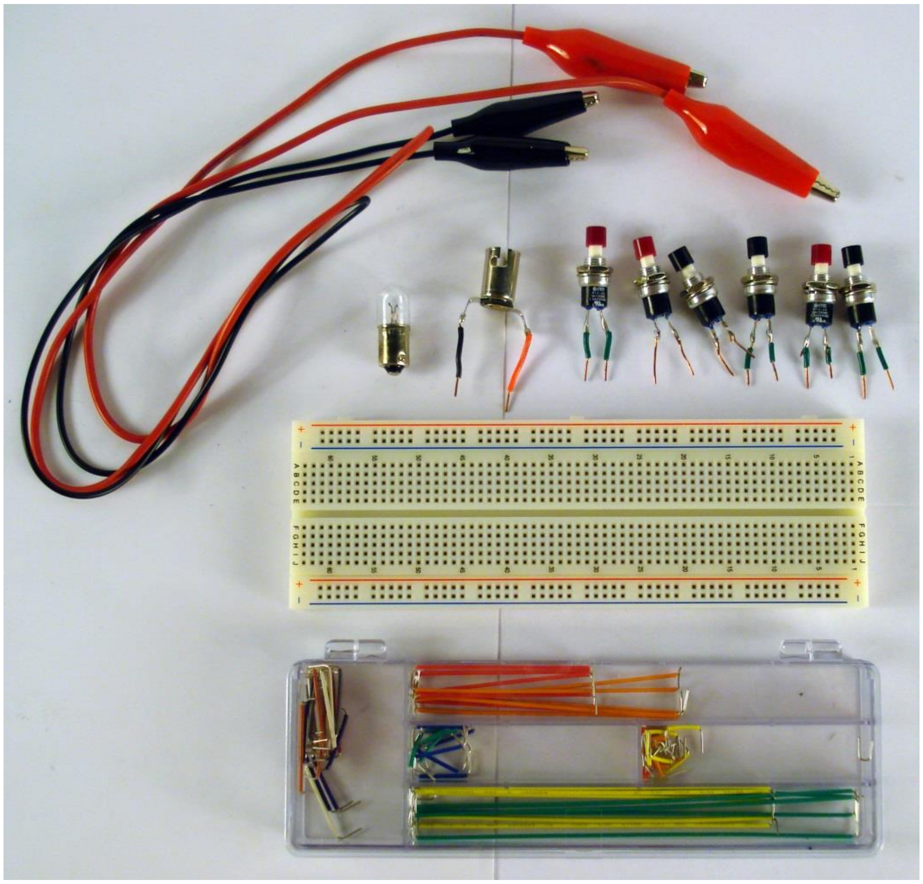
2) Tools for Overview (Section 1):

- Flashing LEDS Learn Soldering kit (for Our First Soldering Activity)
- Tools for building prototypes (see below)
- Digital Multimeter (Pro50A) (see tools for assembly below)

3) Tools for Building Prototypes (Section 2):

These tools are available from Exploring Robotics in a separate toolkit:

- Solderless Breadboard
- Electronics parts (for building Our First Circuit and 6 switches)
- Red and Black (or other colors) wires - separate pieces of various lengths. Or spools of wire and a wire stripping tool
- Red and Black Alligator clips

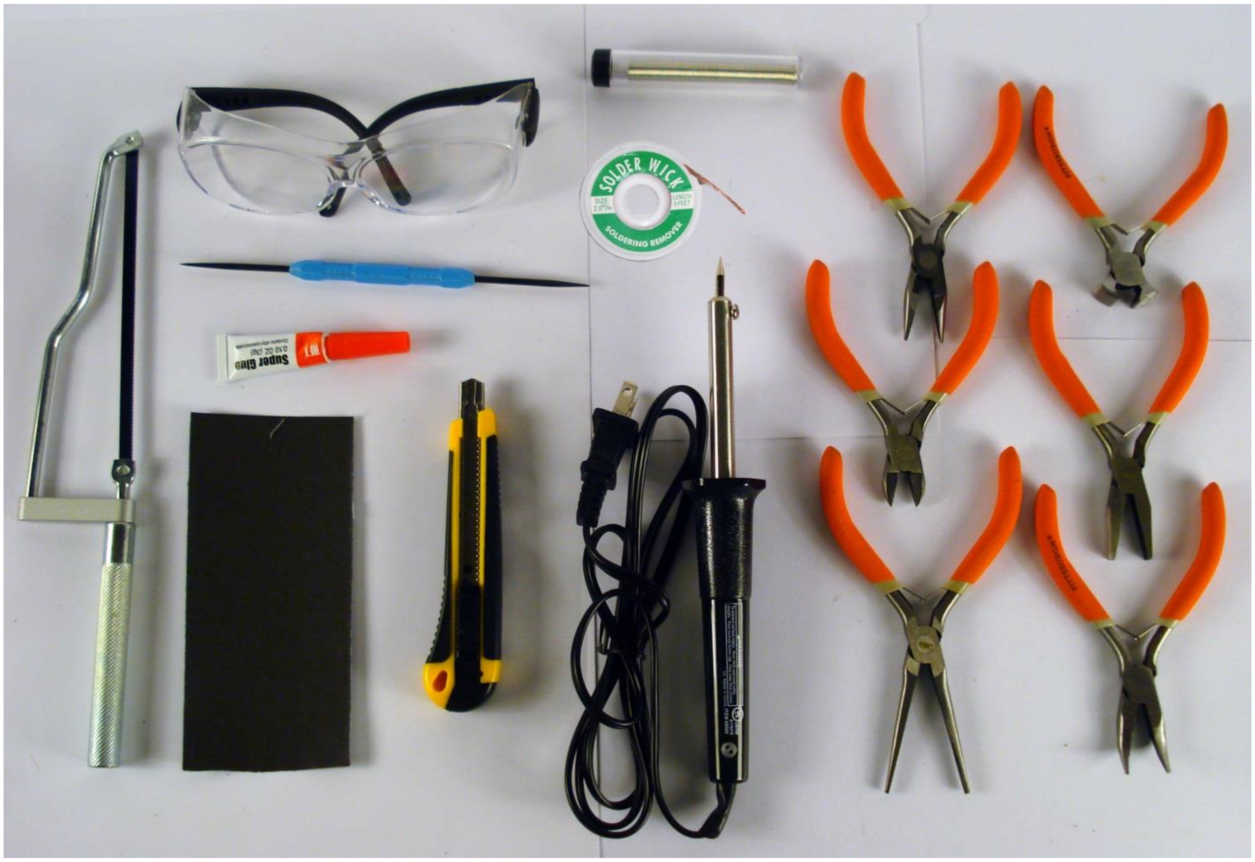


Parts for building breadboard prototypes

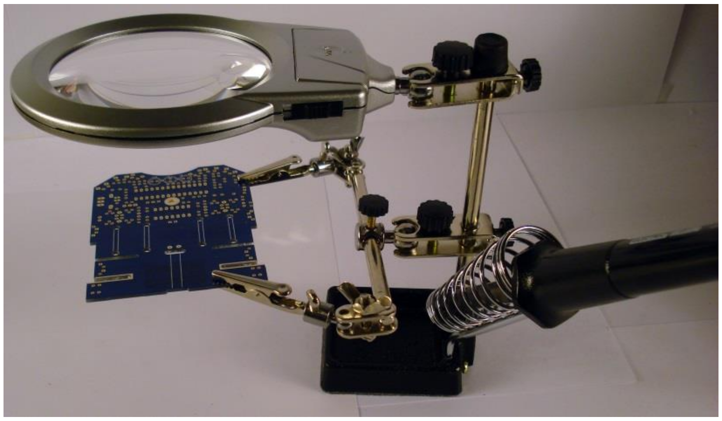
4) Electronics Tools and Parts for Assembly (Section 3):

These tools are available from Exploring Robotics in a separate toolkit:

- A Digital Multimeter
- A “third hand“ with magnifying glass to hold circuit board and see small parts and holder for soldering iron
- Magnifying glass (if not included with third hand)
- Short nose Pliers or Diagonal Cutters for electronic cable work
- Needle nose Pliers for electronic cable work
- Soldering iron for electronic work (approx. 20-30 W) or a soldering station
- Solderwire (1 mm) for electronic work, optionally leadfree
- Desoldering wick (2-3 mm width), just in case solder has to be removed
- Small Hand saw to cut ping pong ball
- Stanley-Knife (xacto)
- Sandpaper (for fine work, Nr. 240)
- Glue (either superglue or a hot glue pistol)
- Sponge for soldering iron
- Reamer tool



Electronics tools required for assembly and prototyping.



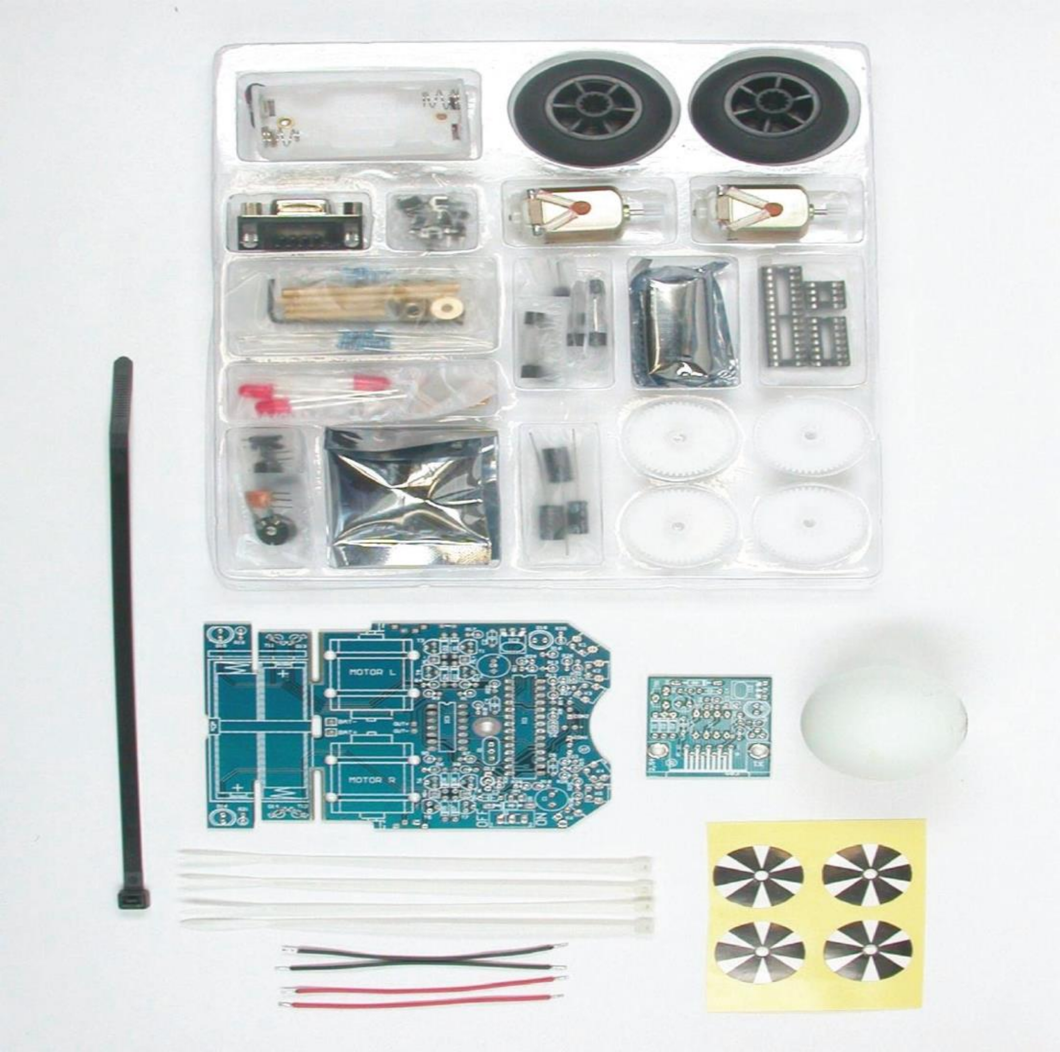
“Third hand” with magnifying glass and solder station.

PRO-50A Digital Multimeter

2. The ARX ASURO Kit

A. Taking Inventory

Before beginning the assembly process it is good engineering practice to count and assure that all parts needed are available. Reading the list of components and using photos to locate them is the first step in learning about electronic components.



B. Parts List

In this list of parts, which come with the ARX ASURO robot, the number with the x is the quantity, for example 2x means that you should have two of them.

Mechanical Parts:

- A table-tennis ball
- 1x Printed circuit board ASURO
- 2x Motors Type Igarashi 2025-02
- 1x Switch (main on/off)
- 1x Battery holder
- 1x Battery clip
- 1x Jumper
- 1x Jumper pins 2pole RM 2.5
- 2x Cogwheel 10/50 cogs; 3.1mm drilling hole Module 0.5
- 2x Cogwheel 12/50 cogs; 3.1mm drilling hole Module 0.5
- 2x Pinion gear 10 (oder 12) cogs; drilling hole 1.9mm Module 0.5
- 2x Collar for 3mm-axle
- 4x Cable binder
- 1x Cable binder releasable
- 2x Rubber tires 38mm
- 2x Messing shaft 42mm long 3mm diameter,
- 2x Messing shaft 24,5mm lang, 3mm diameter
- 2x Encoder sticker

Electronic Parts:

- ca. 15 cm wire red 0.14mm_
- ca. 15 cm wire black 0.14mm_
- 6x Detector switch
- *Integrated Circuits and Sockets:*
  - 1x Processor ATmega 8L-8PC IC (preprogrammed microcontroller)
  - 1x Integrated circuit CD 4081BE
  - 1x IR-receiver SFH 5110-36
  - 3x IC Sockets 14 pol.
- *Diodes:*
  - 1x Diode 1N4001
  - 8x Diodes 1N4148
- *Transistors:*
  - 4x Transistors BC 327/40 or BC 328/40
  - 4x Transistors BC 337/40 or BC 338/40
  - 2x phototransistors SFH300
  - 2x Side-phototransistors LPT80A
- *LEDs:*

  - 1x IR-LED SFH415-U
  - 3x LEDs 5mm red bright diffuse or asymmetric wide-angle
  - 1x Dual-LED 3mm
  - 2x Side-LEDs IRL80A
- *Capacitors:*
  - 2x Elco 220µF at least 10V RM 3.5/10
  - 4x ceramic capacitors 100nF RM 5.08
  - 2x ceramic capacitors 4.7nF RM 2.54
  - 1x Ceramic Resonator 8MHz
- *Resistors:*
  - 1x *coil L1 10uH green (brown, black, black, silver)*
  - 2x 220Ω 1/4 W 5% (red, red, brown, gold)
  - 4x 470Ω 1/4 W 5% (yellow, violet, brown, gold)
  - 10x 1KΩ 1/4 W 5% (brown, black, black, brown, brown)
  - 1x 1KΩ 1/4 W 1% (brown, black, black, brown, brown)
  - 3x 2KΩ 1/4 W 1% (grey, red, black, brown, brown)
  - 2x 4.7KΩ 1/4 W 5% (yellow, violet, red, gold)
  - 1x 8.2KΩ 1/4 W 1% (grey, red, black, brown, brown)
  - 1x 10KΩ 1/4 W 1% (brown, black, black, red, brown)
  - 1x 12kKΩ 1/4 W 1% (brown, red, black, red, brown)
  - 1x 16KΩ 1/4 W 1% (brown, blue, black, red, brown)
  - 1x 20KΩ 1/4 W 5% (red, black, orange, gold)
  - 1x 33kΩ 1/4 W 1% (orange, orange, black, red, brown)
  - 1x 68KΩ 1/4 W 1% (blue, grey, black, red, brown)
  - 1x 1MΩ 1/4 W 5% (brown, black, green, gold)

C. Resistor Color Codes

We can group resistors by their value and identify the value of resistors by the color bands. We look those up using the color chart. The chart in the ARX ASURO manual is a little confusing because it provides both U.S. and European color codes for each resistor.

The four band color codes listed last are the ones that are shipped with the ARX ASURO kit in the U.S. We will refer to those throughout this material. Also note that the gold or silver stripes (for tolerance level) are always on the bottom or to the right when reading resistor colors.

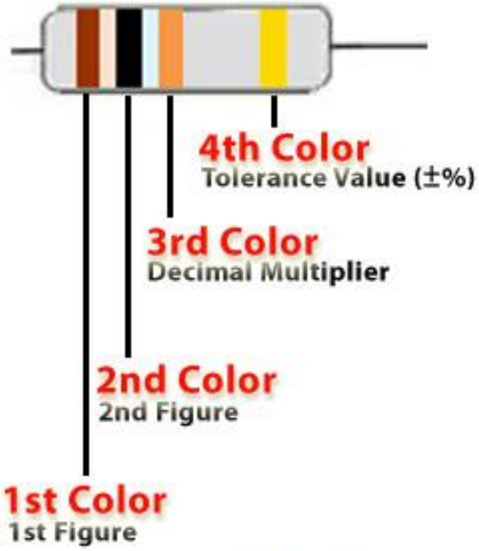
For example,

- We see in the parts list this: 2x 220Ω 1/4 W 5%
- The 2x means there is a quantity of two of them
- The symbol Ω stands for Ohms
- In the color chart, look for 220 Ohm and see that it is red, red, brown, gold

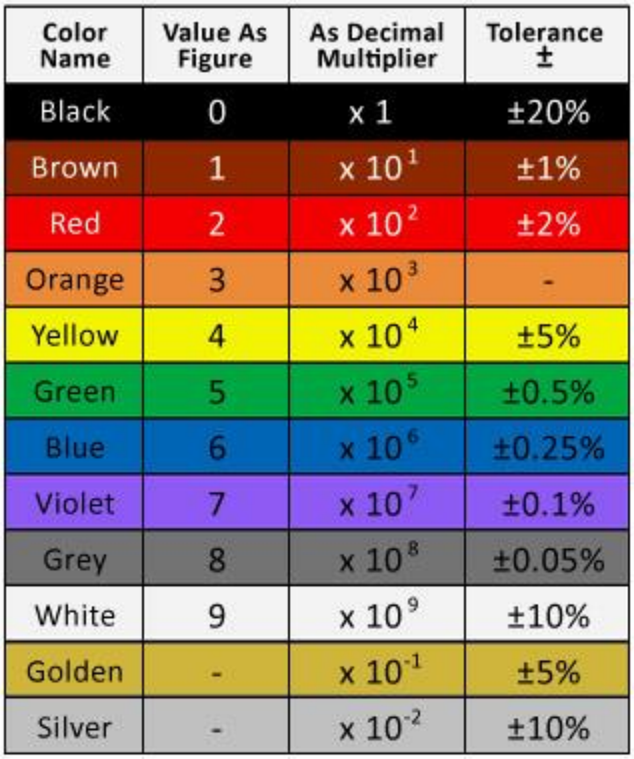
- The 5% tells us the tolerance level which is a Gold stripe
- In the pack of the resistors we find one that looks like this:

Example: 220 Ohm resistor with 5% tolerance

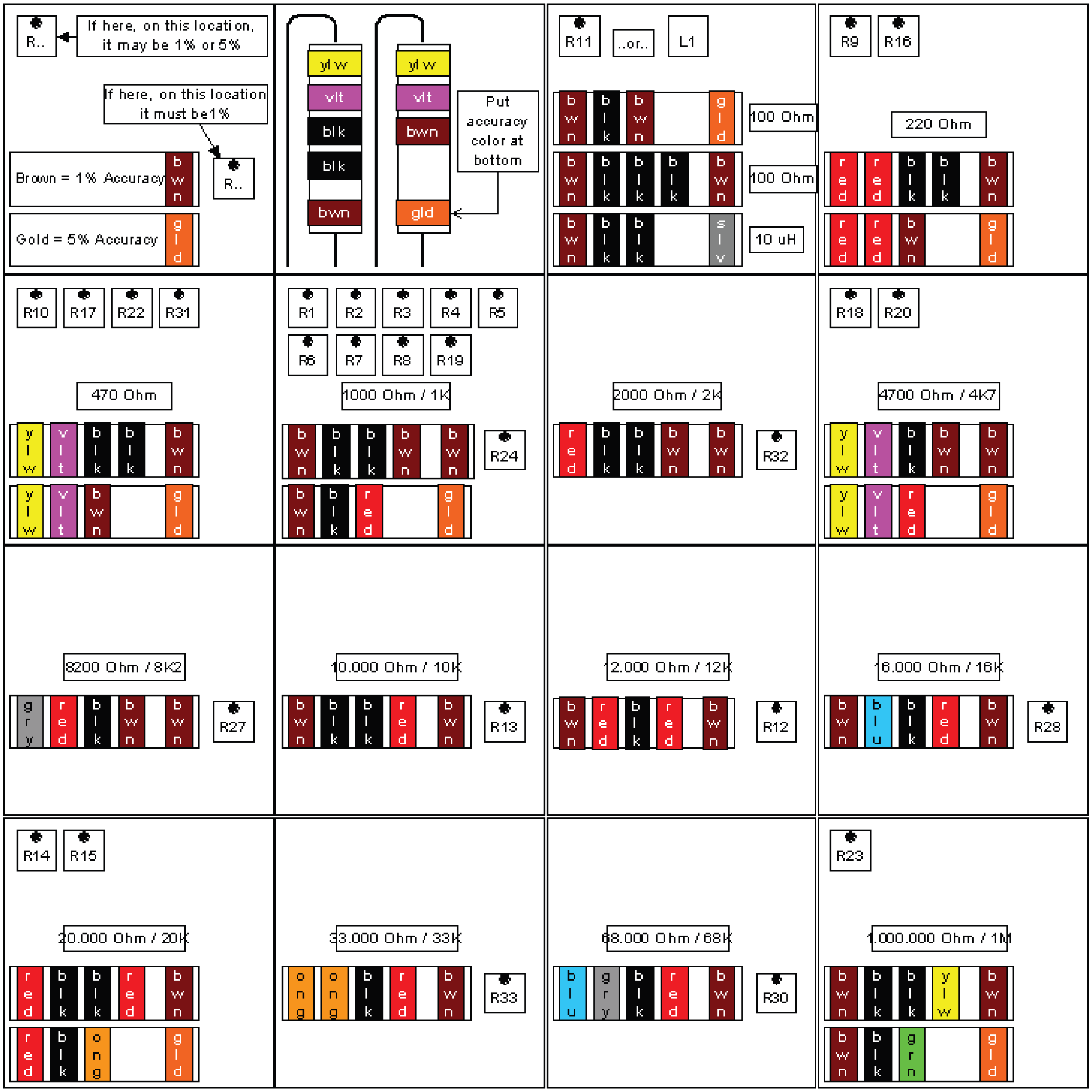
Here is the method used for identifying resistor colors when we know the value:



Above shown resistor's colors are Brown, Black, Orange and Golden so its value is 10 X 1000 = 10000Ω or 10KΩ with a tolerance of ±5%



Color Name	Value As Figure	As Decimal Multiplier	Tolerance ±
Black	0	x 1	±20%
Brown	1	x 10^{1}	±1%
Red	2	x 10^{2}	±2%
Orange	3	x 10^{3}	-
Yellow	4	x 10^{4}	±5%
Green	5	x 10^{5}	±0.5%
Blue	6	x 10^{6}	±0.25%
Violet	7	x 10^{7}	±0.1%
Grey	8	x 10^{8}	±0.05%
White	9	x 10^{9}	±10%
Golden	-	x 10^{-1}	±5%
Silver	-	x 10^{-2}	±10%



Resistor Color Code Chart in the ARX ASURO Manual

3. Getting to Know the ARX ASURO

A. Mechanical Operations

How is the ARX ASURO designed to function? Why was it designed this way?

1) Introduction to ARX ASURO

We are going to build a robot from the ground up. This means that we will look at the design and construction process as if we did not have a solution at hand but rather we are using a design process to create the robot. We do not have time to consider many of the possible ways to build a robot or the possible solutions to each design problem that we encounter. However we will explore several design solutions during our study.

2) Focus on Subsystems

As part of taking a design approach to instruction, we will shift from the approach followed in the ARX ASURO manual. Rather than assembling the entire robot at once, we will instead identify and define each individual subsystem as a solution to a specific problem or task to be accomplished.

We will build a prototype of each subsystem on a solder-less breadboard. There are activities that involve taking measurements using a multi-meter to view how the subsystem actually functions. We will record measurements and document findings to share with an instructor or others.

After we have built all the prototypes, then we will assemble the PCB one subsystem at a time. As we do this we will take measurements using the multi-meter to verify that the circuit is correct. We will also refer back to the measurements taken in the prototype and compare them.

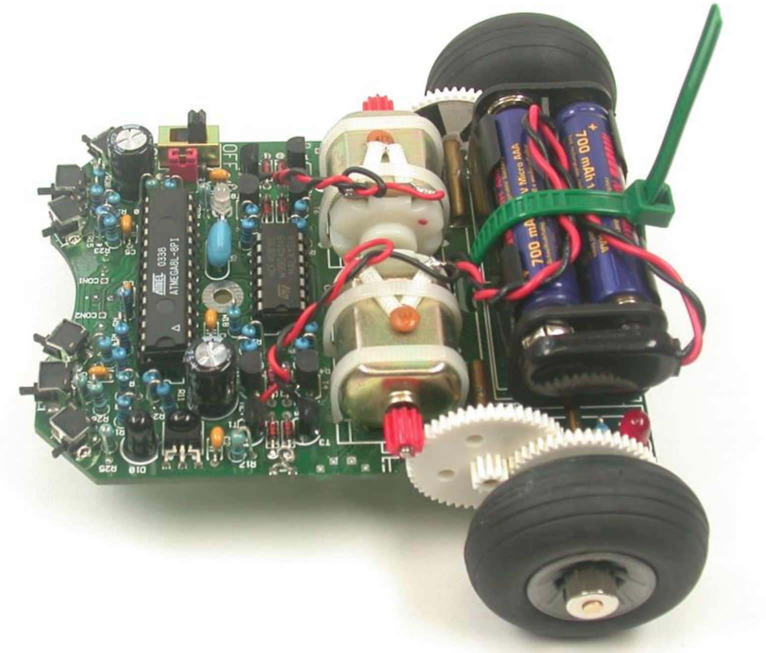
This process will solidify the operation of each subsystem and assure that after all subsystems are developed and assembled, the final robot is likely to operate properly. Also if there are any issues they should be found and corrected during the prototype, assembly, and testing of each subsystem rather than attempting to debug the entire board at one time.

3) Mechanical Operations

The ARX ASURO is designed to move, follow lines, avoid obstacles, operate autonomously, and can be programmed using C language. The physical or mechanical components of the ARX ASURO include:

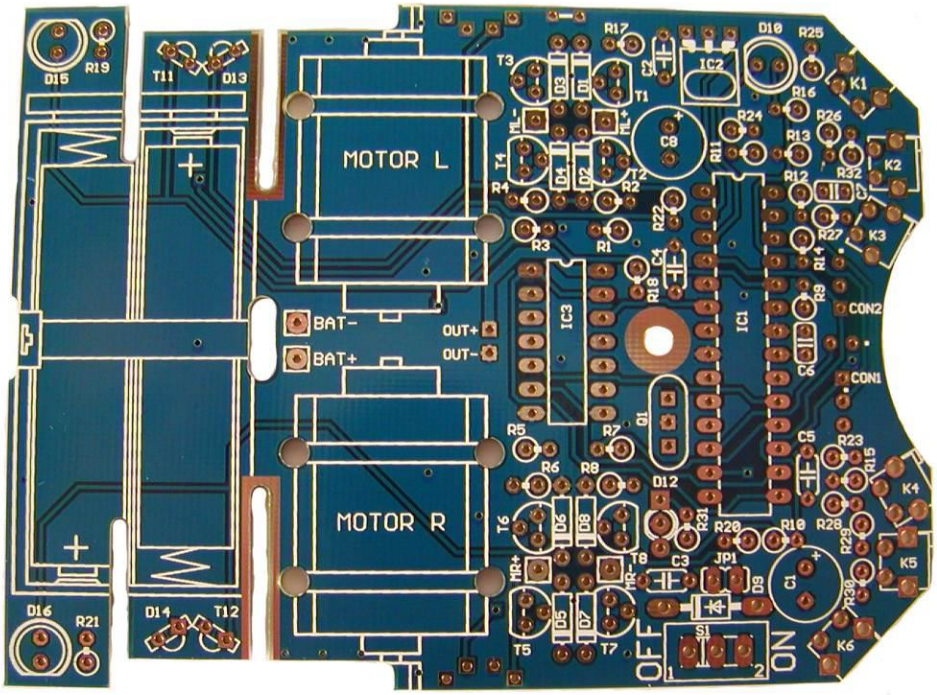
- Body/Chassis – Circuit Board
- Power – Battery Pack
- Wheels and Support/Axles
- Drivetrain
- Gear train
- Motors
- Encoders and Odometer Function
- USB IR-transceiver

Let's go through each of these components so that we become familiar with the parts. We are not yet building the robot, but just getting to know the parts and how the robot operates.



4) Body/ Chassis – Circuit Board

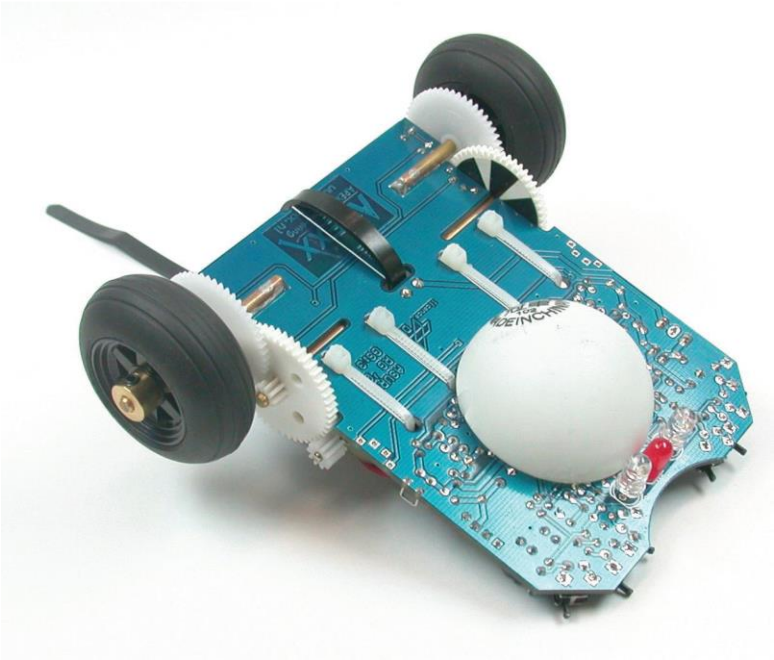
The ARX ASURO is equipped with a RISC processor (microcontroller) on a Printed Circuit Board (PCB) which acts as the robot body or chassis (we tend to use the word chassis as the robot is a type of vehicle). The microcontroller chip, or brains of the robot, is inserted into a socket on the PCB. The components on the circuit board interface with the microcontroller and control or monitor the operation of the mechanical parts. We will be learning about each subsystem, adding the components, and soldering them into the PCB.



5) **Power – Battery Pack**
The battery pack is a plastic housing that tightly holds four (4) AAA batteries in a series configuration. This provides a range of voltage depending upon whether rechargeable or non-rechargeable batteries are used. For rechargeable, the output will be approximately 5V DC and with disposables, the output is approx.6V DC.

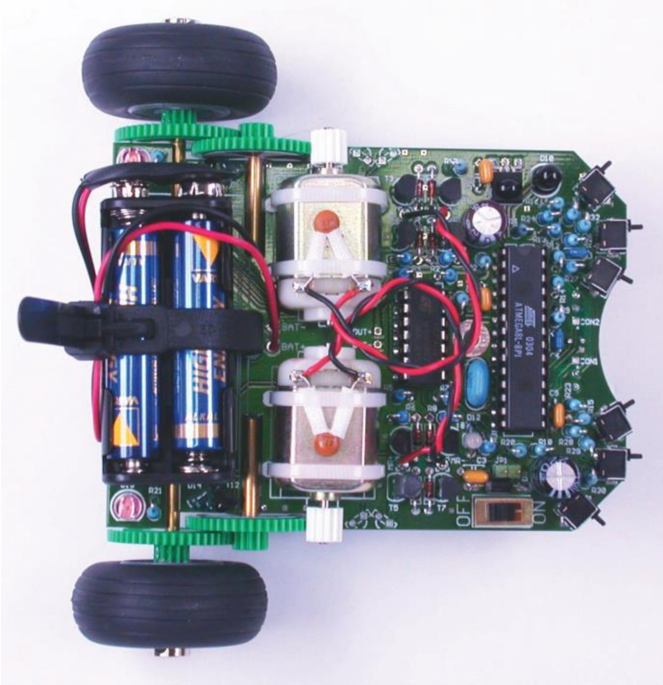
Two power conductors (wires) connect the power supply to the ARX PCB, one red for positive and the other black for negative. The power supply is attached to the PCB board with a plastic tie that can be removed to replace the batteries.

6) **Wheels and Support**
Two motor-driven wheels provide propulsion, or movement for the ARX robot. One wheel is mounted on either side of the ARX chassis. Thus when both wheels are turning to move forward at the same speed then the ARX moves in a straight line. A sliding semi sphere, ½ of a ping pong ball is mounted under the PCB and slides easily on most surfaces thus allowing the driving wheels to pull the chassis around. Steering is accomplished by making one wheel move faster than the other.



7) Drivetrain

The robot drive train consists of the two driven wheels which are connected via gears to two drive motors. There are two gears which combine to reduce motor speed and provide good tractive torque at the driven wheels.



8) Gear Train or Gear Box

The gear train consists of two identical gears arranged in a dual-stage configuration on two axles. This provides transformation of rotational speed or torque from the motors to the wheels via the small gear that is attached to the motor.

9) Motors

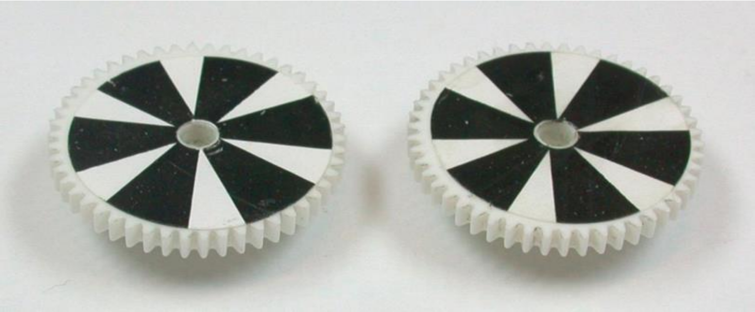
ARX's motors are designed to operate at around 3 to 7 VDC, with DC batteries. They are brushed DC motors. The simple electric motors contain a turnable armature, a fixed permanent magnet, and a commutator. The armature uses three coils and is located between the permanent magnet's poles. *(If you don't understand these terms, see the resources for an overview of how DC motors work.)*

The specifications of the motor are:

- Voltage range: 3.0 to 12.0 V
- Speed range: 12,400 to17,500 RpM
- Current range: 0.12 to 0.34 Amps
- Torque: 0.10 Ncm
- Power output: 1.26 W
- Efficiency: 30%
- Weight: 18 g

10) Encoders and Odometer

One of our large gears has an encoder label attached to its interior side. This encoder label is shown here. For each rotation of the gear, the black spots on the label will generate six pulses from the phototransistor which monitors transitions from light to dark of the encoder label. The count of the pulses is used to determine the distance traveled.



11) USB IR-transceiver

A communication link is provided by the IR transceiver circuit consisting of a USB IR transceiver connected to a remote PC and an IR receiver and IR transmitter LED which make up the ARX portion of the communication link.

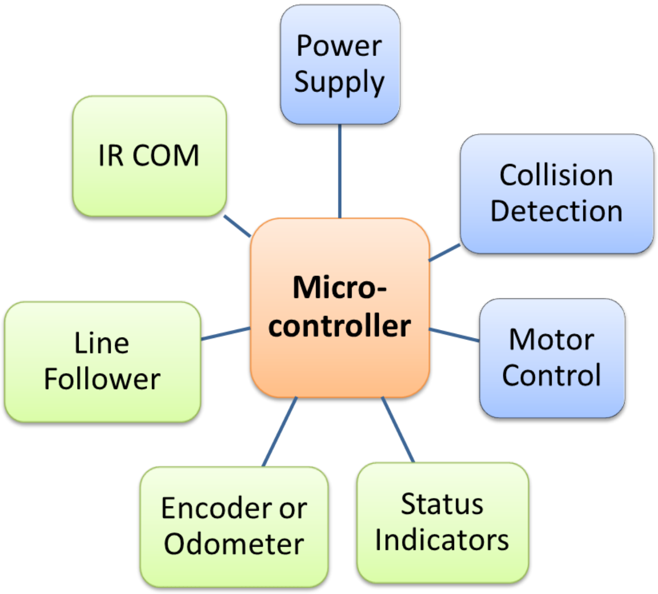
B. ARX ASURO Functions

The ARX ASURO robot is made up of subsystems, each controlling a primary function of the robot. We will be building one subsystem at a time and then testing its operation.

1) Overview of Systems

Let's get more familiar with the subsystems by going through a system level block diagram. With this we can get an upper level view so that we can see how all the parts work together and create a mental framework of the overall operation of the robot. The primary functions or subsystems of the ARX ASURO are:

- Microcontroller
- Power Supply
- Collision Detection Circuit
- Motor Control Circuit
- Status Circuit
- Encoder or Odometer Circuit
- Line Follower Circuit
- IR COM Circuit



2) Microcontroller

Microcontrollers are a special type of computer (similar to the Mac or PC) but the difference is that the microcontroller can have all that it needs to make decisions on a single device. That means that it contains memory for storing programs, memory for storing data, decision making circuits, analog signal processing circuits, inputs and outputs for sensors and control of external devices, and other functions all on one chip that plugs into a printed circuit board. It is considered the brains of the robot, and all other devices connect to and communicate with it. This allows the robot to sense things about its environment and then make decisions and take action, which makes it seem almost alive or smart.

3) **Power Supply**

The ARX power supply is a relatively simple device consisting of a battery holder that contains four batteries wired in series which provides approximately 6 volts DC power, and an on-off switch. There is also a power supply circuit containing components which assure that the power supply voltage level is adequate for operation.

4) **Collision Detection**

One set of devices that are critical for navigation are the collision detection switches. These six switches are mounted on the front of the robot and provide information to the microcontroller when the robot bumps into an object. The microcontroller can take different actions based upon which switch or set of switches sense a collision.

5) **Motor Control**

Two motors provide tractive force to move and steer the ARX robot. The two motors drive a set of gears which in turn the drive wheels to cause motion. The microcontroller can control both the speed and direction of each motor. These motor control functions are derived from the motor control circuit also called an H-bridge. The devices that make up this circuit controls on/off, speed, and direction of rotation of the motors.

6) **Light Activated Subsystems**

The ARX uses four light related subsystems for operation and control. We will consider each of these subsystems independently, but for now we will provide an overview of their operation and what they have in common.

- Operation Status Indicators
- Line following
- Odometer or Encoder
- IR communications

7) Operation Monitoring And Status Indicators

The status LED circuits are the simplest of the light related subsystems. The ARX robot uses LEDs to indicate proper operation of various subsystems. The LEDs allow us to determine when the ARX is operating properly. The microcontroller monitors operation and turns on or off the status LEDS to communicate with the human operator that the system is operating properly or that something has gone wrong. These status LEDs are shown in the table.
LED #, Purpose, Status, Port.

8) Odometer or Encoder

The ARX manual refers to the encoder circuit as an odometer. The logic behind this name is that the encoder circuit senses the presence or absence of a black target mounted on one of the ARX gears. As the motor turns the gear, the light and dark areas of the target are sensed with a phototransistor. Counting the number of transitions from light to dark provides monitoring of the rotation of the gear and thus the relative position of the ARX can be determined programmatically and calculate the distance traveled - hence the name odometer. We prefer using the term encoder as this term has far broader use in many industrial applications.

9) Line following

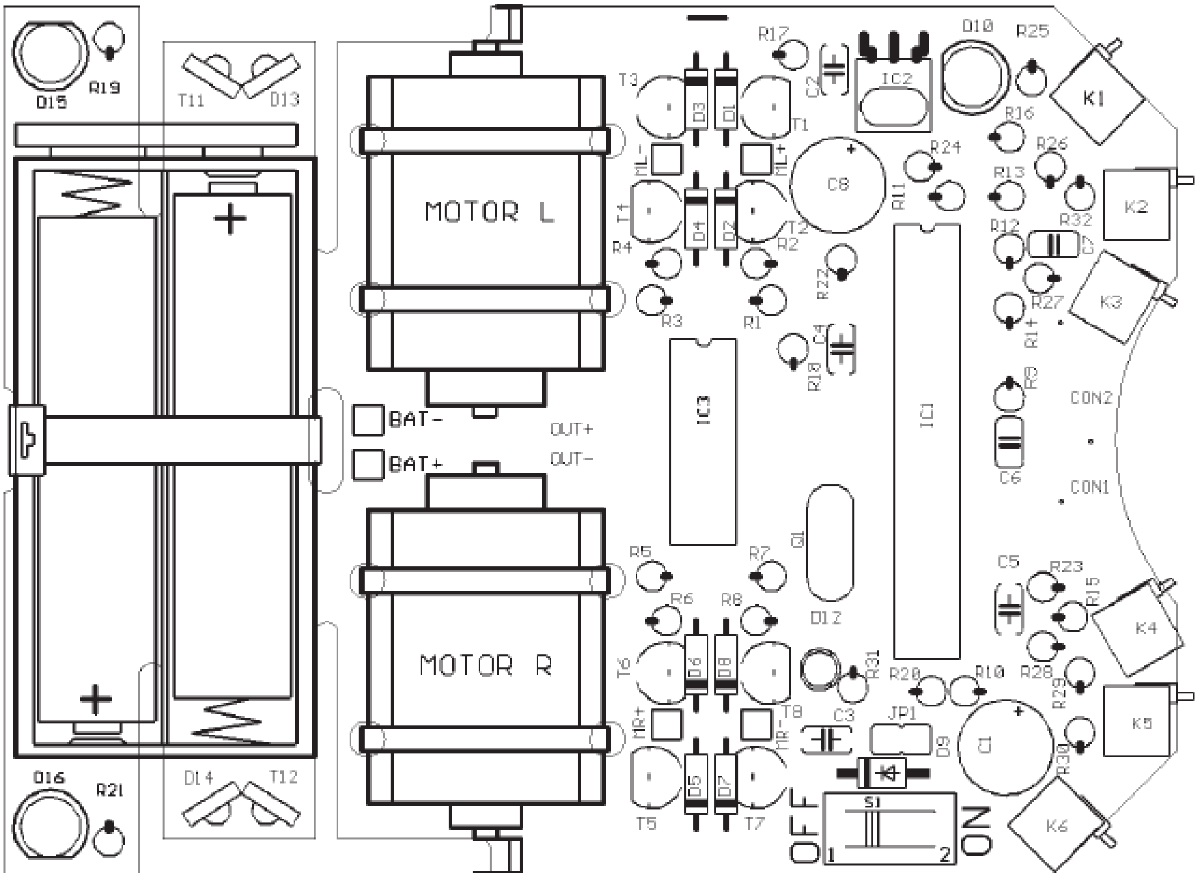
Line following is enabled by using three devices working together. An LED illuminates the line and two phototransistors sense reflected light to determine the presence or absence of the line. A high contrast black line against a white background is the line to be followed. In the motor control we learned how transistors work and were able to control the robot motors by using transistor switches.

For line following, we use another type of transistor - a phototransistor. This device does not require an external base current supply, but rather the transistor is turned on via a light signal. We will build a prototype of this light sensitive line tracing circuit and monitor its operation using a meter.

10) IR Communication

The IR Communication subsystem consists of three main elements and some passive components. IR signals sent from an IR equipped PC computer are sensed by a IR semiconductor device. This special sensor receives on-off pulses at a defined frequency. The pulses, like the on and off of Morse code are transferred to the microcontroller where they are interpreted and acted upon. IR communication can be used to download programs to the microcontroller where they are executed or run. The IR can also be used to give specific commands to cause the robot to take pre-defined actions. This is a less costly way of communicating with the PC than using BlueTooth or a USB cable connection.

Two way communication with a remote PC is obtained by using an infrared LED to transmit information from the ARX to the remote PC. This two way communication closes the communication loop and allows the ARX to also report information regarding its operation back to the PC. We will build a prototype of the IR communication system and use a meter to monitor some of its functions.



4. The Microcontroller Brain

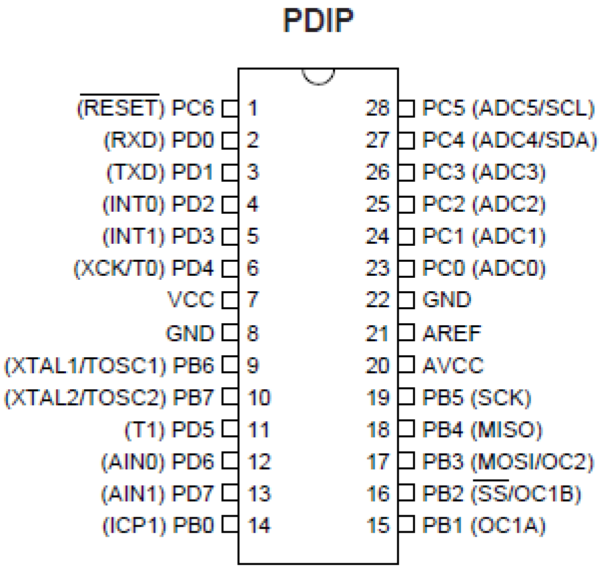
Microcontrollers are amazing tiny decision making devices that control many of our everyday things such as: phones, microwaves, televisions, lighting, sprinkler systems, garage doors, keyboards, and automobiles. In fact an automobile may have from 50 to over 100 microcontrollers controlling everything from power windows and seats to temperature.

Microcontrollers are in traffic signal lights, dishwashers, clothes washers and dryers, stoves, toaster ovens, in other words almost every modern appliance we use has a microcontroller making our lives easier and more convenient. But perhaps the most important use as far as this course is concerned is that a microcontroller is used to make decisions for a robot. Also the digital Multimeter that you will use to take electrical measurements also has a microcontroller brain.

A. The ARX ASURO Microcontroller

The ARX ASURO microcontroller Integrated Circuit (IC) consists of a black oblong plastic part which contains very tiny electronic circuits with 28 metal legs extended underneath. The metal legs fit into a socket which is placed on the PCB board. The legs connect the microcontroller chip to other devices.

There is an indent in one end of the chip which helps to orient it correctly into the socket. When looking down from the top, the number 1 pin is to the left of the indent. Pins are numbers 1 to 14 on the left side and 15 to 28 on the right side of the IC.



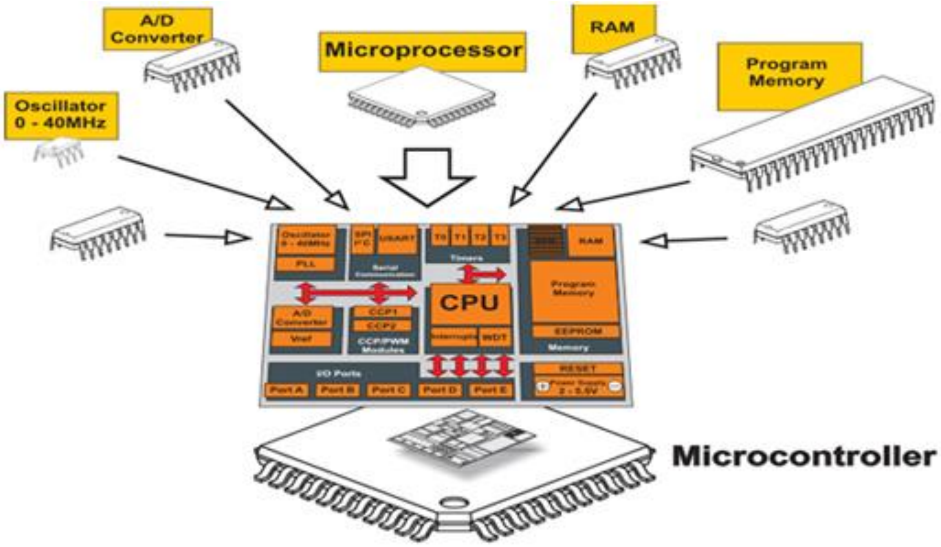
1) A Closer Look at the Microcontroller

OK so these are great little decision making devices but how do they work? Hundreds of books have been written on that subject. You can attend university courses on the topic, get a graduate degree in it, and spend a career working with the subject.

You can travel the world working on projects about how to use and apply microcontrollers. You can go into space, travel to the moon or planets thanks to microcontrollers. But still we must answer the question how does a microcontroller do its thing? We can provide only an overview.

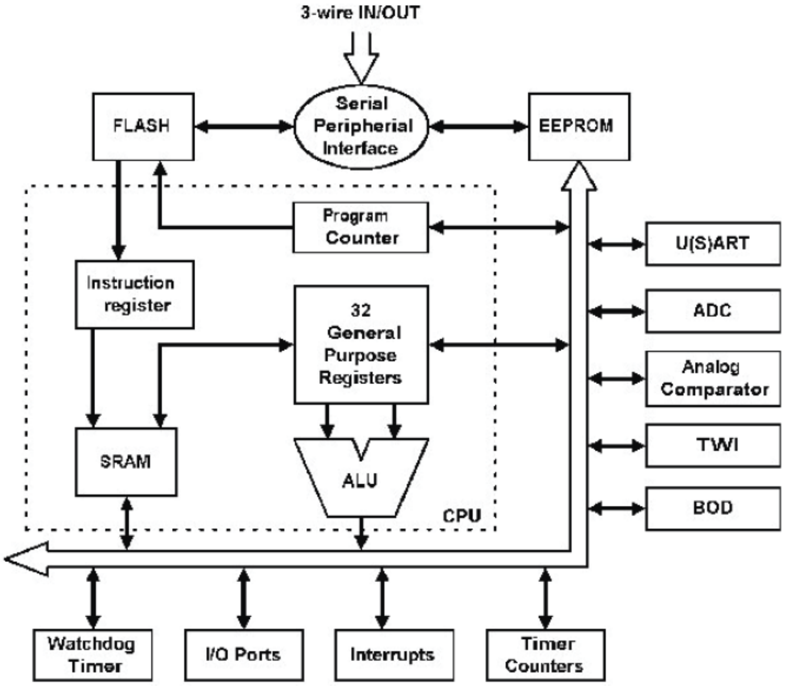
2) Microcontrollers are Computers

Microcontrollers are a special type of computer (similar to a microprocessor in a Mac or PC) but the difference is that the microcontroller can have all that it needs to make decisions on a single device. That means that is contains its memory for storing programs, memory for storing data, decision making circuits, analog signal processing circuits, inputs and outputs for sensors and control of external devices, all on one chip that plugs into a printed circuit board.



General overview of what a microcontroller contains. Each chip is unique. See manufacturer's specifications for details.

B. Block Diagram Functions



This is a block diagram of the microcontroller, which shows the functions inside the microcontroller chip. We don't need to understand these functions to be able to program or control the robot. But let's point out what a few of the terms mean.

1) Central Processing Unit

Notice the dotted line indicates the functions that are part of the CPU or Central Processing Unit. Those include the Instruction register, Program counter, General Purpose Registers, SRAM, and ALU.

- The instruction register holds the program code that is currently being executed.
- The program counter holds the location of the program code. It keeps track of which line of code is being run during execution of the program.
- The General purpose registers hold the current values of variables and data that is being processed.
- ALU stands for Arithmetic Logic Unit and it handles the math computations for the programs.
- SRAM stands for Static Ram. This is a type of memory used for storing data. This system has 1,000 bytes of SRAM.
- Flash is a type of memory used for storing programs. This microcontroller has 8,000 bytes of in-system programmable Flash memory.

2) Other Functions of the Microcontroller

- The large arrow with other boxes connected is called the bus. A bus is a method of transporting data between the CPU and I/o devices; it is a collection of wires carrying information with a common purpose.
- EEPROM stands for Electrically Erasable Programmable Read-Only Memory is also a type of memory. This chip has 512 bytes of EEPROM. This is used to store permanent data on the chip.
- The Serial Peripheral Interface allows communication with serial devices such as Infrared or connection to other types of serial links.
- The Watchdog timer is a method for checking the proper operation of the chip. It does so by having a timeout which is monitored and reset periodically.
- The I/O Ports are the general purpose Input Output ports. These allow data to be input and output to devices.
- Interrupts are a special type of input that will interrupt the normal program and run special high-priority code.
- Timer Counters are general purpose counters and timers used for control purposes. They are used to provide special timing or count functions to time events and have things which occur programmatically at specific times.
- U(S)ART are special communication channels for communicating with special devices such as other microcontrollers, computers, or specialized chips.
- ADC is an Analog to Digital Converter that allows voltage signals to be read from sensor devices to obtain data such as speed, battery voltage, phototransistors, line following, or switches used for collision detection.
- Analog Comparator is used to compare an external voltage to a known reference. This is used with the ADC.
- TWI is a Two-wire Serial Interface. The TWI protocol can interconnect up to 128 different devices using only two bi-directional bus lines, one for clock (SCL) and one for data (SDA).
- BOD is a Brown-out Detector. This device monitors the voltage input to the microcontroller and provides an alarm function if the voltage drops below a pre-set limit. If the voltage drops below that point, the microcontroller becomes unreliable.

3) Features of the ATMega8 Microcontroller:

The ATmega8 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. The ATmega8 provides the following features:

- 8K bytes of In-System Programmable Flash with Read-While-Write capabilities,
- 512 bytes of EEPROM,
- 1K byte of SRAM,

- 23 general purpose I/O lines,
- 32 general purpose working registers,
- three flexible Timer/Counters with compare modes,
- internal and external interrupts,
- a serial programmable USART,
- a byte oriented Two-wire Serial Interface,
- a 6-channel ADC where four channels have 10-bit accuracy and two channels have 8-bit accuracy,
- a programmable Watchdog Timer with Internal Oscillator,
- an SPI serial port, and
- five software selectable power saving modes.

The ATmega8 AVR is supported with a full suite of program and system development tools, including C compilers, macro assemblers, program debugger/simulators, In-Circuit Emulators, and evaluation kits.

Five software selectable power saving modes:

i. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning.
ii. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next Interrupt or Hardware Reset.
iii. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.
iv. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions.
v. In Standby mode, the crystal/ resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption.

C. Binary Number System

Everything a microcontroller does is based upon binary logic or ones and zeroes (and the same is true of a computer). I am writing this on a PC laptop, in a program called Word, but for the computer to understand what I am doing it first converts all my actions (typing) to information in the form of ones and zeroes.

This can be done with a microcontroller watching the keyboard keys. As a key is pressed the microcontroller decides which key it is and then sends a pattern of ones and zeroes over a pair of wires to the PC computer processor. The processor has loaded an application named Word which is currently in control, and takes the action of displaying what I typed (or doing something else) based upon the program code in the Word ap.

If we look at a keyboard there are letter keys, number keys and a lot of control keys. Each key can have a different meaning for our Word Ap. A Cap Lock key can mean that all subsequent letter keys pressed will be represented as capital letters. The special END key can tell Word to move to the end of the current string of characters being typed.

Using a microcontroller to monitor and control keyboard operations reduces what must be done by the main computer processor and also reduces cost, speeds up processing, and makes it possible to produce millions of keyboard processing microcontrollers that are used in a huge number of laptops and desktops, and other devices.

1) Welcome to Our Crazy Binary World

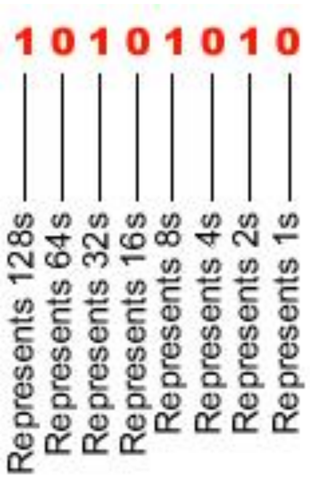
Understanding how this microcontroller does this allows us to understand how computers understand anything in the world or our entire universe!! That sounds either crazy or magic; to be able to understand everything based upon binary which boils down to just ones and zeroes.

You are about to discover that it is neither crazy nor magic, but simply clever. Let’s return to our keyboard microcontroller, we have separate number keys, and we can type any number from 0 to any value we want, so how does the computer know what that number is if it can only understand ones and zeroes. It does the trick by converting our digital number to binary numbers. Let’s look closer at binary.

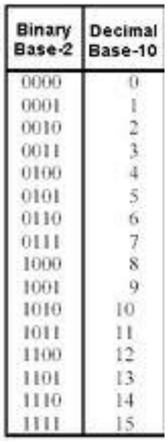
2) What is this 11?

OK if you said eleven, then you can get what binary numbers are because you automatically know that we have one ten plus one more to get the number eleven. We say our number system is based on 10. Some people say it is based on tens, because we have ten fingers. What if we came from an alien planet and only had two fingers? How would we then count with just two choices? We have a way of doing just that and it is called a binary, meaning two number system.

Here is a link to a video at the Kahn academy. Kahn does a great job of explaining the binary system. *As a matter of fact, Kahn does a great job of explaining many topics in Math and science. Keep it in mind whenever you have a question about something technical.* Here is the link: https://www.khanacademy.org/science/computer-science-subject/computer-science/v/binary-numbers



OK if you viewed the video you now know how to represent the beginning of numbers in binary format. We started with 2 and progressed through 2 to the 2nd power which is 4, and 2 to the 3rd power or 8.
As this chart shows we go from right to left with each digit starting at the ones place, then the twos place, then the fours, and next the eights place. So with a four digit binary number we can count up to 15 things.



Binary Base-2	Decimal Base-10
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15

- 1 thing is binary 0001
- 3 things is binary 0011
- 6 things is binary 0110
- 9 things is binary 1001
- 15 things is binary 1111

But we often have more than 15 things. What do you think comes next? If you guessed 16 you are correct. 2 to the 4th power is 16. After 16 comes what? 32. And after 32? 64. And after 64? 128. So that is eight digits.

If we continue on to nine digits, after 128 we reach? If you guessed 256 you are right. So each time we move to the next level we are multiplying by two (2). So if we continue the process we have 512, and then 1,024, and then 2,048, and then 4,096.

So when we have 8 digits in a binary number, we can count up to 255 things. And when we have 12 digits in a binary number, we can count up to 4,095 things. We could continue to increase forever but for now we will stop and return to our keyboard and its microcontroller.

3) 10 Bits for Analog to Digital Convertor

Our microcontroller uses a 10 bit resolution for the Analog to Digital Converter when any sensed voltage is converted to a number. So what does that mean? It means that a measurement is taken and the result of that measurement is a 10 digit binary number. Thus it can be divided into 1,024 parts (or we can round it to approximately 1,000 parts).

Let's consider our water tank. Suppose we had a thousand gallon tank, and our level monitoring gave us a 10 volt signal when full, and a 0 volt signal when empty. Then if our full range input is divided by approximately 1,000 parts of resolution, this would mean that we can sense a one gallon difference in the water level (1,000/1,000 = 1).

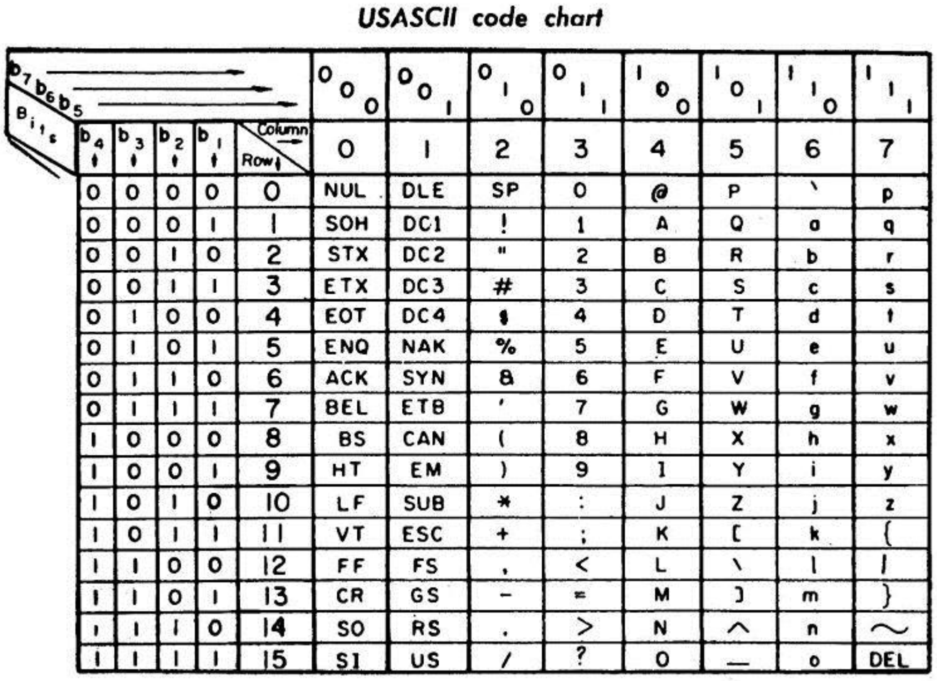
Imagine next that we had a 5,000 gallon tank. In this case the minimum shift in level that we could sense would be 5,000 divided by 1,000 parts which means that the minimum that we can sense is a 5 gallon change in level: (5000/1000) = 5 gallons.

4) ASCII Code

Our keyboard microcontroller can handle any numbers we enter by converting them to their binary equivalent. But what about all those letters and punctuation marks and control keys? Take a guess.

If you guessed that we will convert those keys to binary you are right. But before we get to Binary, we first convert each key to its ASCII number or the American Standard Code for Information Interchange. There is a table of ASCII codes in the appendix.

Wikipedia is a good reference for some types of information and in this case does a pretty good job of describing how letters and control keys are converted into binary format. Here is the link http://en.wikipedia.org/wiki/ASCII. You do not need to understand this in depth but simply know that everything we deal with in a microcontroller is converted into binary: ones and zeroes.



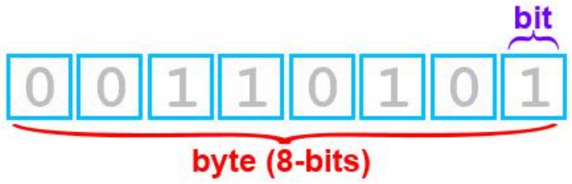
USASCII code chart

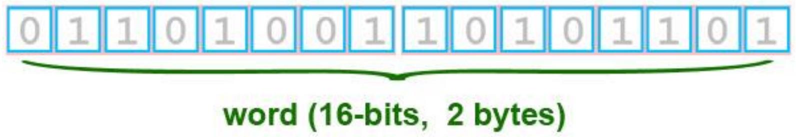
b7 b6 b5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
Bits b4	b3	b2	b1	Column → Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	—	o	DEL

5) Memory Registers

Another place where the binary system is used is when storing information into memory. We see in the microcontroller block diagram that there are several types of memory in a microcontroller.

The measurement unit for memory is bytes. Each place in a binary number is called a bit. A four-bit number is called a nibble, an eight-bit number is called a byte. So a byte is a unit of data that is eight binary digits long. A byte is the length needed to represent a character such as a letter, number, or typographic symbol (for example, "h", "2", or "?"). Larger numbers are called "words" which may be 16, 32, 64 or 128 bits long. This is not to be confused with the “words” used in a sentence. They are not the same thing.

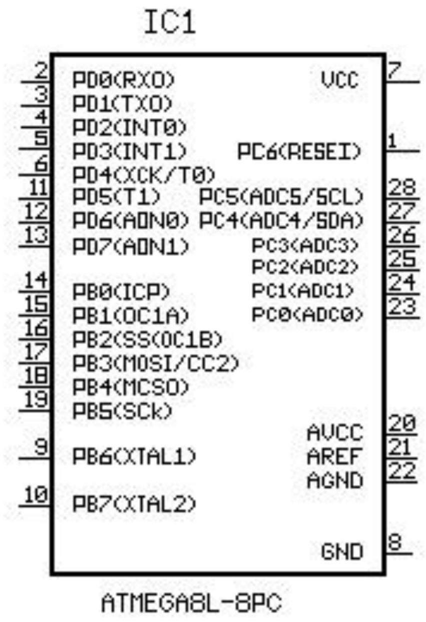




So when there are 1,000 bytes of memory, that means the memory can store approximately 1,000 characters. Microcontrollers usually have less memory then PC computers. They don’t usually process pictures or documents or data that requires larger amounts of memory.

D. Communications

The program loaded onto the microcontroller receives signals from devices on the PCB board through pins which are then referenced as ports in a program. We will see that as we build circuits, each circuit is connected to particular pins on the microcontroller. Later when we get into programming, we will see how the programs interface with the ports.



1) Pins - How the Microcontroller Communicates

This diagram shows the pin numbers around the outside and the associated port numbers inside the rectangle which is representing the IC chip. The pin numbers are shown above the line which represents a connection point to the chip. They are placed in an order which made it easier to draw the schematic.

It is important to note that this diagram (which is used in all the circuit diagrams) does not correlate to the physical location of the pins. The physical location of the pins are numbers 1 to 14 on the left and 15 to 28 on the right. The number 1 pin is to the left of the indentation on the chip which helps with proper orientation when inserting the chip into the socket. So to find the physical location of pin 5, start with pin 1 and count up 4 more pins.

The particular microcontroller chip we are using on the ARX ASURO is an ATMEGA8L-8PC.   Each microcontroller has its own definition of pins and ports. So the ones we use on this robot will be different than ones used on a robot that has a different microcontroller.

The signal received by the microcontroller is a binary signal of ones and zeros because it is receiving an electrical signal from various electronic devices in a circuit. Electricity is either on or off, current is either flowing or not flowing. But then various programming commands may convert the binary number into an ASCII number or a decimal number. The program logic uses the number as information to determine what action to take.

We will get more into this when we start to program the ARX ASURO. But for now, it is important to know that the microcontroller only uses binary numbers, but when we write a program, we can use binary, ASCII, Hexadecimal, or decimal numbers.

2) Analog to Digital Converter

Analog voltages tell us more than if something is on or off, they may also allow us to tell “how much”. Imagine we want to monitor the water level in a tank. In order to do so we may want to know that the tank is empty, and also that it is full. Both of these conditions are important, to say the least. However, we may also want to know how full.

Let’s say that the tank holds 10000 gallons. If we only monitor the two conditions full and empty then we only know if the tank is not empty and not full. Imagine that the tank has only 20 gallons left. Do you think this is a critical piece of information? Most of us would answer yes. In order to measure information of this type we can use an analog sensor.

An analog sensor provides a signal which is proportional to a desired piece of information, such as how many gallons of water are in the tank. It might not only be able to let us know that we only have 20 gallons remaining but also when we have say 110, 200, 1500, or 9950 gallons available. Isn't this better than just knowing that the tank is either empty or full?

The device we use to measure the water level provides information in an electrical current which is considered an analog signal. What we need is a way to convert that analog signal to a number that our microcontroller can understand, a device that converts analog signals to digital numbers. So that is where the name came from for the device we want to use: an analog to digital convertor, (or ADC).

The ARX ASURO robot has a few sensors which provide analog signals for functions that support the encoder, the line follower, and battery voltage. These use the ADC to convert their signal to a number.

E. Integrated Circuits

When computers were first designed they depended upon vacuum tubes to provide the logic required for decision making. Vacuum tubes are a special type of electronic device that has a relatively simple form of operation. Inside of a glass tube, a wire is heated. When the wire becomes hot enough electrons are emitted into the vacuum space. Another device called a plate - simply a metal plate - is located near the heated wire. The plate is charged with a positive charge to a power supply and the wire or cathode is connected to the negative side of the power supply. Electrons move across the vacuum and are collected on the plate.

A third device called a grid is made of screen like metal and is located between the heated wire and the plate. When a negative potential (wire connected to negative side of battery) is placed on the grid, it tends to keep the electrons from flowing from the wire to the plate. Thus by controlling the amount negative potential, we control the flow of electrons. The grid becomes the controlling device. The electrons can move through the openings in the grid, but they only do so when it is less negatively charged, when it becomes positively charged it repels the electrons and keeps them away from the plate. This provides an on and off state that is controllable with pulses of electricity and allows circuits to be built to create the logic of a computer.

The first computers were built using vacuum tubes, although they had many issues or problems. Vacuums had many negative issues. First, they were large and second they required a large number of them to build a computer. If you have a smart phone, you have more computing power on it than a computer that filled an entire room made of vacuum tubes.

Next they required heat to operate. Remember we heated the wire called the cathode in the tube to initiate the flow of electrons. This required a lot of electricity and meant that there was excess heat. Cooling a large number of tubes became an issue.

Next, they were not very reliable. The computer is large, very complex, and unreliable. Vacuum tubes burn out - the wire erodes as electrons are given out and eventually fails. Because of the large number of tubes required, replacing burned out tubes became an on-going chore.

1) A Small Solution - Transistors

Often when a technical problem occurs, bright minds come up with a solution. In this case a solution was small when compared to the vacuum tube. It was the transistor. A transistor is small, does not depend upon heat to operate, therefore it did not require as much power or cooling. They also last much longer than vacuum tubes, so they require less maintenance. They still required a large number of transistors to build a computer, but they took up less space and were far more reliable. Transistors were the first major leap forward in computing.

*"**Transistors** are devices that control the movement of electrons, and consequently, electricity. They work something like a water faucet -- not only do they start and stop the flow of a current, but they also control the amount of the current. With electricity, transistors can both switch or amplify electronic signals.*

*The transistors made at Bell Labs were initially made from the element germanium. Scientists there knew pure germanium was a good insulator. But adding impurities (a process called **doping**) changed the germanium into a weak conductor, or **semiconductor**. Semiconductors are materials that have properties in-between insulators and conductors, allowing electrical conductivity in varying degrees."*
(Source: http://electronics.howstuffworks.com/transistor1.htm)

2) Integrated Circuits - Another Giant Step Forward

Not long after transistors were developed, engineers again worked to increase reliability, decrease size, and increase the speed of operation. This occurred with integrated circuits. An integrated circuit is made up of many transistors mounted and wired on a single device.

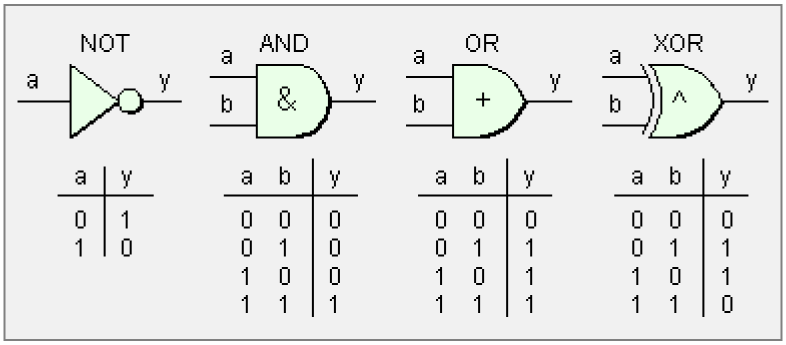
3) Millions of Transistors

Once integrated circuits were designed, they continued to develop rapidly with ever increasing numbers of transistors. Moore's law states that each generation of integrated circuits will double the number of transistors, increase the speed of operation, and decrease the size and cost of the integrated circuit.

4) Logic Gates

A logic gate has an output which depends upon two or more inputs. As an example, an AND gate has an output of 1 if both inputs are 1. If only one of the inputs is 1, then the output is 0. We show this relationship with a truth table. A truth table defines the possible states of inputs and outputs for a logic device. The table below shows the symbols for

AND, OR, NOT, and XOR. The inputs are often represented by A and B and in this table, the output is y.



a	y
0	1
1	0

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

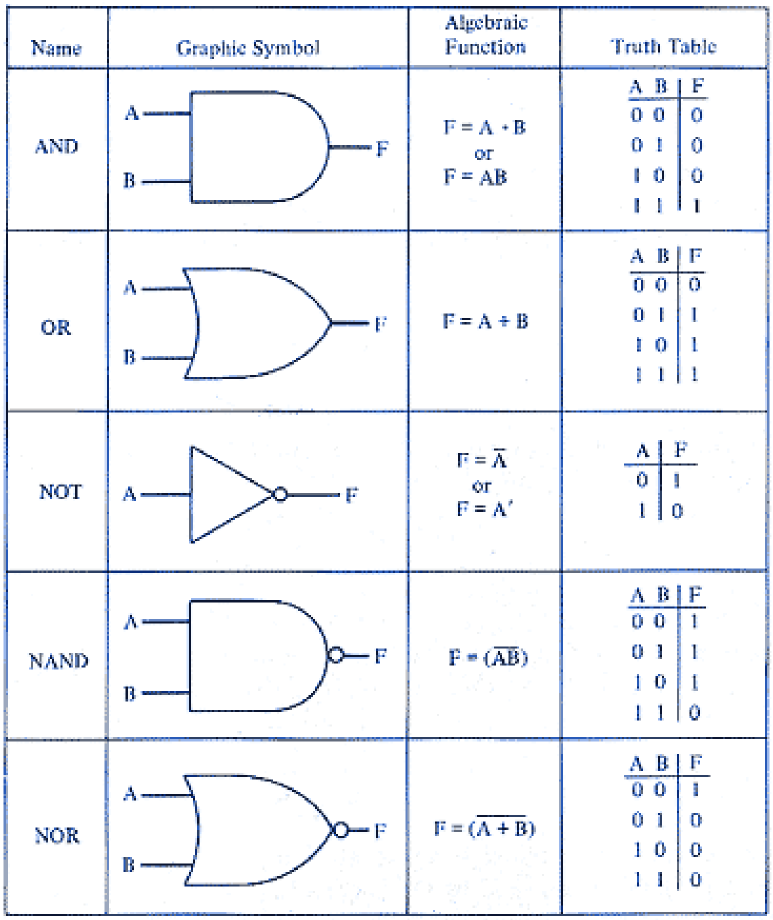
a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

An OR gate logic is that if either A or B equals 1, then y is 1. If A and B both equal 1, y will be 1. If A and B are both equal to 0, then y will be 0.

An XOR or exclusive OR gate logic is that if A is 1 and y is 0 then y is 1. If A and B are 1, then y is 0. y is only 1 when either A or B are 1, but not when both are.

Binary Logic Table

This table summarizes the various functions available with logic gates.



Name	Graphic Symbol	Algebraic Function	Truth Table
AND	A B F	$F = A \cdot B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR	A B F	$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT	A F	$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND	A B F	$F = (\overline{AB})$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR	A B F	$F = \overline{(A + B)}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

At the fundamental chip level, logic boils down to a few basic logic elements which work with binary data. These are AND, OR, NOT, NAND, NOR, and XOR. You may recall something similar from algebraic logic functions.

In the graphic symbols shown here, the inputs are represented by the lines on the left labeled “A” and “B” and the output is the line on the right labeled “F”. Recall that when we are using voltage and current in a circuit, that the current is either flowing which is a 1, or it is not flowing which is a 0.

An AND gate has two or more inputs and a single output. It produces an output of 1 when ALL of its inputs are 1. When we look at the truth table, there are other combinations of inputs of 0 or 1, and those all result in an output of 0. This logic is “If A AND B are 1, then F is 1”.

An OR gate has two or more inputs and a single output. It produces an output of 1 when ANY of its inputs are 1. When we look at the truth table, there are other combinations of inputs of 1, and those all result in an output of 1. When both inputs are 0 then the output is 0. This logic is “If A OR B are 1, then F is 1.”

A NOT gate performs an inversion, and is sometimes called an Inverter. It outputs the opposite of the input. When the input is 0, the output is 1. When the input is 1, the output is 0. This logic is "if A is 1, then F is 0, or if A is 0, then F is 1."

A NAND gate stands for Not AND. It is a combination of an inverter and an AND gate. When both inputs are 0 then the output is 1. When both inputs are 1 then the output is 0. When either input is 1 then the output is 1. When you compare the truth table to the AND gate, basically we take the output of the AND gate and invert it.

A NOR gate stands for Not OR. It is a combination of an inverter and an OR gate. When both inputs are 1 then the output is 0. When both inputs are 0 then the output is 1. When either input is 1 then the output is 0. When you compare the truth table to the OR gate, basically we take the output of the OR gate and invert it.

5) ESD Electrostatic Discharge

All electrical and electronic components use electricity. But unfortunately certain components are very sensitive to too much electrical charge. The most common way in which components are damaged is through static electricity, called ESD or electrostatic damage. Static electricity is caused by rubbing too dissimilar materials together. An example is to walk across a carpet and touch a doorknob and receive an electric shock. Although that shock is not great enough to injure you, it is sufficient to damage many electronic components, especially Integrated Circuits (ICs).

It is common practice for technicians and engineers to make sure there is no static charge when touching electronic components. This can be accomplished by using a grounding strap connected to the wrist which safely discharges any static charge. If a grounding strap is not available, then discharge static electricity by touching a device that is grounded such as a metal radiator or a metal water pipe. In most instances, the metal case of a computer is grounded electrically, thus touching the case will normally eliminate any static charge.

NOTE: Processor chips: IC1 ATmega, IC3 CD4081and the IR-receiver IC2 SFH5110-36 are sensitive to ESD (electrostatic damage). They may be damaged by simply touching or even reaching for these parts if you have been charged with electricity (eg. by walking on a carpet).
Before handling these parts you should discharge yourself with a discharging bracelet, connected to ground, or at least by touching a grounded heater system or a metallic case of equipment.

F. ARX ASURO Logic Gates

Logic gates are also a special type of electronic component in the form of an integrated circuit (IC). They are similar in construction to a microcontroller IC but have fewer legs or leads and are smaller is size.

The AND logic gate used in the ARX ASURO has 14 pins. It performs an important fail safe function for the motor control circuit. It prevents us from accidently creating a direct short across the power supply, which would result in burning out the power supply.

The ARX ASURO Robot uses a logic gate to obtain a control function. It uses an AND gate where a 1 is a 5 volt signal, and a 0 is a 0 volt signal in the motor control H bridge. We will learn more about the motor control circuit later.

With the development of micro-controllers, there are fewer reasons to use individual logic gates, however, from time to time they can come in handy to perform a special function, as is the case with the motor control circuit of the ARX ASURO. Here it provides a safety mechanism.

5. Engineering Skills

A. The Engineering Design Process

The engineering design process involves a series of steps that lead to the development of a new product or system. The engineer (or student) should be able to do the following:

STEP 1: Identify the Problem. State the challenge problem in your own words. Example: How can I design a __________ that will __________?

STEP 2: Identify Criteria and Constraints - Specify the design requirements (criteria). List the limits on the design due to available resources and the environment (constraints).

STEP 3: Brainstorm Possible Solutions - Sketch ideas as a group discusses ways to solve the problem. Labels and arrows should be included to identify parts and how they might move. These drawings should be quick and brief.

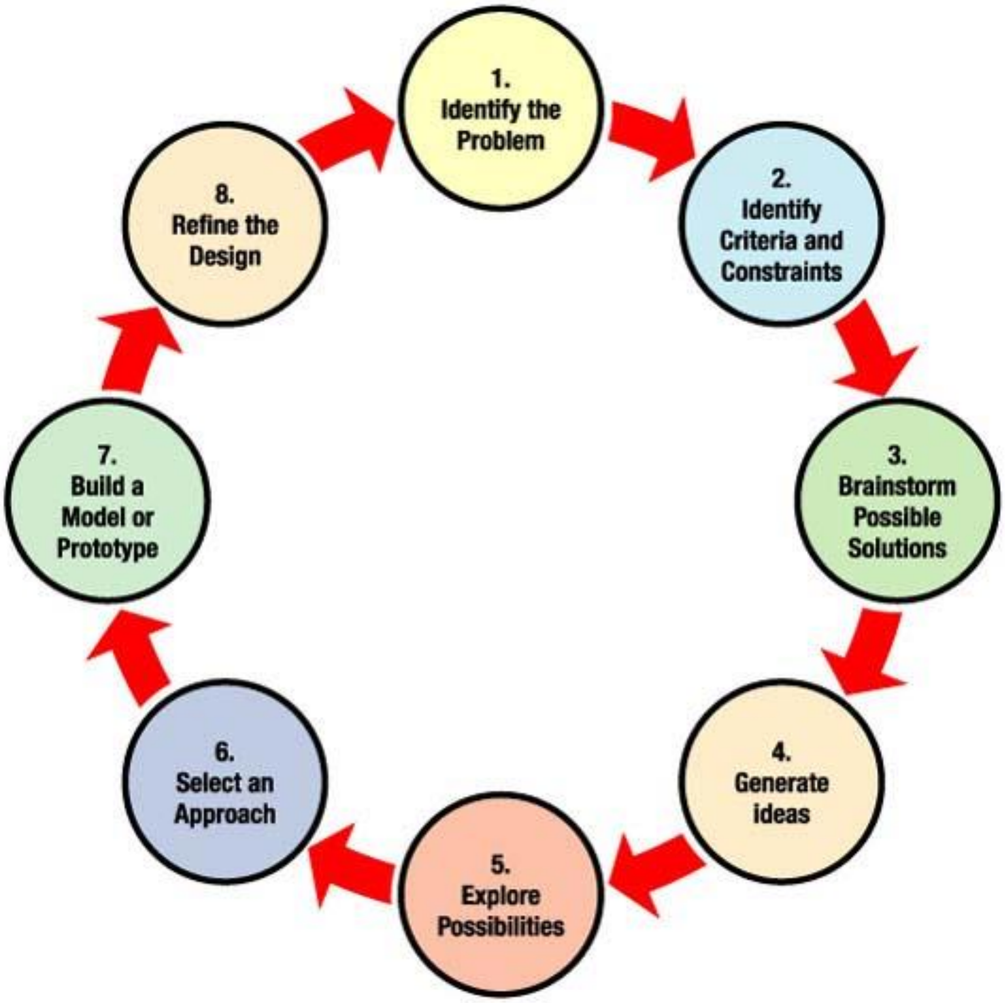
STEP 4: Generate Ideas - Develop two or three ideas more thoroughly. Create new drawings that are orthographic projections (multiple views showing the top, front and one side) and isometric drawings (three-dimensional depiction). These are to be drawn neatly, either using a CAD system or using rulers to draw straight lines and to make parts proportional. Parts and measurements should be labeled clearly.

STEP 5: Explore Possibilities - The developed ideas are shared and discussed among the team members. Record pros and cons of each design idea. Think outside the box for creative ways to solve problems that arise with meeting criteria or constraints.

STEP 6: Select an Approach - Work in teams and identify the design that appears to solve the problem the best. Write a statement that describes why they chose the solution. This should include some reference to the criteria and constraints identified above.

STEP 7: Build a Model or Prototype - Construct a full-size or scale model based on the drawings. Identify and acquire appropriate modeling materials and tools.

STEP 8: Refine the Design - Examine and evaluate prototypes or designs based on the criteria and constraints. Present to potential customers who review the solution and help identify changes that need to be made. Based on criteria and constraints, teams identify any problems and proposed solutions.



B. ARX ASURO Engineering Design

The ARX ASURO was designed by a team of German engineers as a fun educational robot. Some of the design objectives, criteria, and constraints for the ARX ASURO were to have a robot which is highly responsive, accelerates and decelerates rapidly, turns quickly, and has a high top speed. They also wanted to keep the costs low for educational use. Some of the design choices that were made and the ways used to accomplish this are:

- Keep weight of robot low by not adding additional parts and incorporate the chassis into the PCB board.
- Use small high speed DC motors that are lightweight to control the wheels.
- Reduce speed using gears and also increase torque (but still have a relatively high rpm rate on motors even after reducing with gears).
- Minimize number of electronic components needed for control and thus reduce size of PCB board and weight of robot.
- Use size AAA batteries which are lighter and are smaller while still maintaining a good amount of run time.

- Use alternative combinations of hardware and software to cleverly achieve control objectives which minimize parts and minimize I/O requirements of microcontroller. Examples are in the motor control H bridge circuit and collision detection circuit.
- Should be easy to navigate and follow a preprogrammed path. This is met with using an optical encoder system to facilitate navigation and speed control by incorporating a simple optical encoder mounted on the primary drive gear.
- Provide a method of simple wireless communication between the robot and a controlling PC. Also allow for downloading programs to the robot. To reduce costs and maintain simplicity in design this uses an infrared based system.
- Provide a visual method of indicating proper operation of systems. Lightweight and low cost LEDs are used as status indicators.
- Provide a method of line following that is lightweight and simple to implement. Method chosen is based on infrared LEDs in combination with phototransistors.
- The robot needs to have good traction, good grip, and respond well to rapid acceleration and deceleration. The wheels and tires are made of rubber and are sized to increase speed at a high motor rpm.

C. Engineering Skills

There are many skills that you will develop as you explore a career in any technical field. There is no one single course to acquire all of these skills, nor is there a single college program. Some of the skills require on the job training. But keep this list in mind as you choose particular courses or internships or jobs in your career path. Some job skills that are important to electronics technicians and engineers are:

- **Critical Thinking** - Using logic and reasoning to identify the strengths and weaknesses of alternative solutions, conclusions or approaches to problems. Creating engineering solutions, designs, and prototypes of products, machines, tools, or systems.
- **Troubleshooting** - Determining causes of operating errors and deciding what to do about it.
- **Complex Problem Solving** - Identifying complex problems and reviewing related information to develop and evaluate options and implement solutions.
- **Reading Comprehension** - Understanding written sentences and paragraphs in work related documents. Reading and understanding schematic diagrams, engineering drawings, tables, charts, product descriptions, books, engineering journal articles, and other technical documents.
- **Writing** - Communicating effectively in writing as appropriate for the needs of the audience such as engineering journaling and notebooks, creating presentations, creating specifications, creating training documents, creating reports, and writing journal articles and “white papers”.
- **Working with Machines and Tools** – Using machines and tools to design and build engineering solutions. Repairing machines or systems using the needed tools. Controlling operations of equipment or systems. Determining the kind of tools and equipment needed to do a job. Performing routine maintenance on equipment and determining when and what kind of maintenance is needed.
- **Application of Theoretical Knowledge** - Applying principles and theories of electronics, electrical circuitry, engineering mathematics, electronic and electrical testing, and physics.

1) Electronics Engineering Traits:

Each of us is born with specific personality traits. We then take those traits and enhance them by learning new skills. To be a good electronics engineer, you should have:

- a flair for math, science, technology, and Information Technology
- the ability to analyze problems
- strong decision-making skills
- excellent communication skills
- the ability to prioritize and plan work effectively

- the ability to manage a budget of time and costs
- excellent 'people' and team working skills
- a clear understanding of electrical safety regulations.

2) Industry Use of Schematic Diagrams

One example skill is reading schematic diagrams. Learning to read schematic diagrams is a lifelong process. Engineers and technicians learn new tricks, and have new devices offered to them from companies that specialize in providing solutions to ever more complex problems. Companies like Texas Instruments, National Semiconductor, Honeywell, Cypress Semiconductor, and hundreds of others spend millions of dollars annually developing solutions to problems.

The auto industry is an excellent example of research and development looking for solutions. As government adds restrictions on pollution, it requires that auto manufacturers use microcontrollers and sensors to achieve both better gas mileage as well as pollution reduction. Solution providers develop sensors and microcontrollers that aid mileage and pollution reduction and offer them to the industry. The industry is huge and a solution provider can sell millions of devices.

3) Become a Detective

Sometimes reading a schematic is like being a detective. The first thing to do is to look for clues. If there is an associated operators or owner’s manual then this may be the place to look for clues about how the system is supposed to operate. Knowing how a circuit is to operate gives an idea about many of the components. A toaster oven may have a knob operated switch that allows selection of a function or temperature control. It may also have a keypad that allows input of temperature or time. The owner’s manual will define the operation of the controls which will give clues as to how the circuit works. We are not suggesting that you take a toaster oven apart. *Do not do this as it has high voltages that can shock, injure and even kill!*

4) Do Not Destroy Evidence

A good detective is very careful to examine and collect information, but is careful to never destroy evidence. Don’t tear things apart. Examine carefully, be careful. Use research tools. The Internet is a great source of information. You can find millions of schematics online along with descriptions of how the systems operate. Follow trade journals to learn about an industry. Auto, Medical, aerospace, petrochemical, pharmaceutical, gas exploration, heavy equipment, automation, mining, transportation, telecommunications, robotics, food processing, baking, all have trade journals along with online forums that offer information. Go online and learn what these industries are doing. As you consider a career you can learn a lot about the companies involved in an industry that will help you get a job and after you have a job help you get an advancement.

5) Everyone is the same but Different

Many industries and even companies within and industry may have accepted practices that define how they do things that may differ from more general practices. By studying you can learn what those practices are and that will help you read technical documentation and schematics for that industry.

D. Electronics and Schematic Diagrams

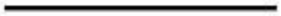
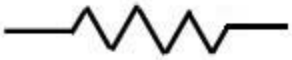
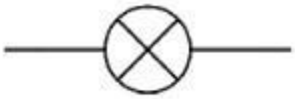
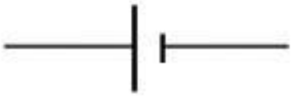
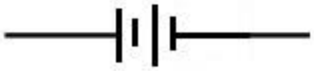
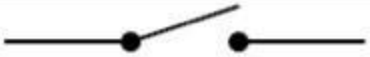
Whenever a device is built that includes electronics, it starts with a schematic diagram. The schematic diagram is the road map which includes all the electronic components and how they are connected together. There are some standards which are followed when creating diagrams. The symbols used are standardized so that technicians and engineers can read the schematic and understand how the device functions.

1) Electronic Parts

There are a large number of electronic components that are used to build all types of equipment and devices. A couple of things they have in common is that they can be represented by an electronic symbol, and they all require electricity. Symbols are used in schematics which illustrate how electronic and electrical components are inter-connected. As we build the ARX ASURO we will learn about many different components and their associated symbols. Technicians and engineers must be able to recognize the symbols and determine the electronic components in any schematic diagram.

2) Symbols Used in Schematics

Each electronic device has an associated symbol. Symbols are easier and faster to draw than the real component. To read a schematic drawing, you must be able to recognize the symbols. Here are a few example symbols for a wire, a resistor, a light bulb, a cell, a battery, and a switch.

Component	Circuit Diagram Symbol
Wire	
Resistor	
Light bulb	
Cell	
Battery	
Switch	

3) Schematics for the ARX ASURO

The overall schematic diagram for the ARX ASURO robot is included below. At first glance this schematic seems very complex, and it is. Rather than attacking it as a whole, we have broken it out into the 8 subsystems for the ARX ASURO. We will examine each subsystem independently as we build the prototypes. This reduces the complexity and helps us to learn the whole process by first learning each subsystem.

We will build a prototype - which is a temporary circuit - of each subsystem using a solder-less breadboard. As we build the prototype we will refer to the schematic diagram because it provides a roadmap for how items are connected. It also allows us to become familiar with each component, its symbol, and how it can be connected to other devices. Testing each subsystem allows us to more fully understand the circuit operation and relate them back to the schematic. With this process we build skills that are required of every technician and engineer.

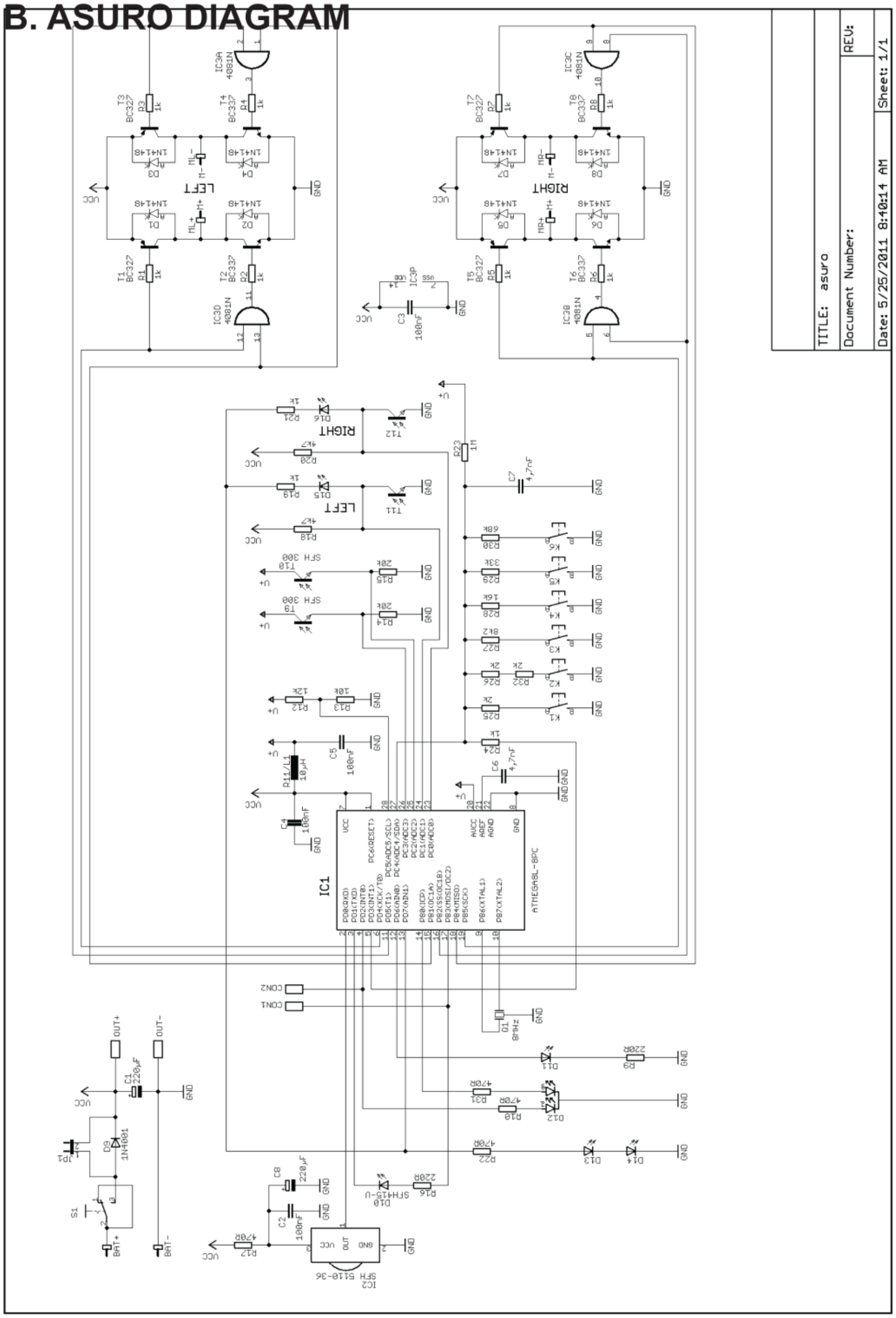
4) Reading Schematic Drawings

Here are a few pointers to remember when reading a schematic drawing:

- Each part of the circuit (capacitor, resistor, connector, transistor, etc.) is represented as a symbol in the diagram. Normally each symbol is labeled by a characterizing letter code, a sequential part number in the schematics (eg. R1, C1, D3), and the part's measurement value.

- Lines symbolize the connection nets (wires, cables, or printed circuit board wires) between part entries.
- Line crossings with a dot symbolize making connections between nets, whereas line crossings without a dot symbolize separate nets (or wires).
- Sometimes special nets, e.g. ground or supply voltages use special symbols instead of a number of lines, greatly improving the readability of the schematic diagram. Just imagine all identical net symbols with the same terminal labels to be interconnected by lines.
- Negative (Black wire) connections are indicated as GND or VSS or V-.
- Positive (Red wire) connections are indicated as VCC, or VDD or V+.
- Integrated circuits - a module containing a number of discrete elements on a semiconductor chip - and other devices often will be drawn as graphical symbols (such as rectangles) symbolizing the schematic function, with a description or part number.
- Schematics which incorporate integrated circuits or ICs often do not correlate the pin locations to physical locations on the IC. They will show the pin locations in the drawing, but put them in an order which is most convenient for creating the drawing – trying not to cross wires more than necessary.

B. ASURO DIAGRAM



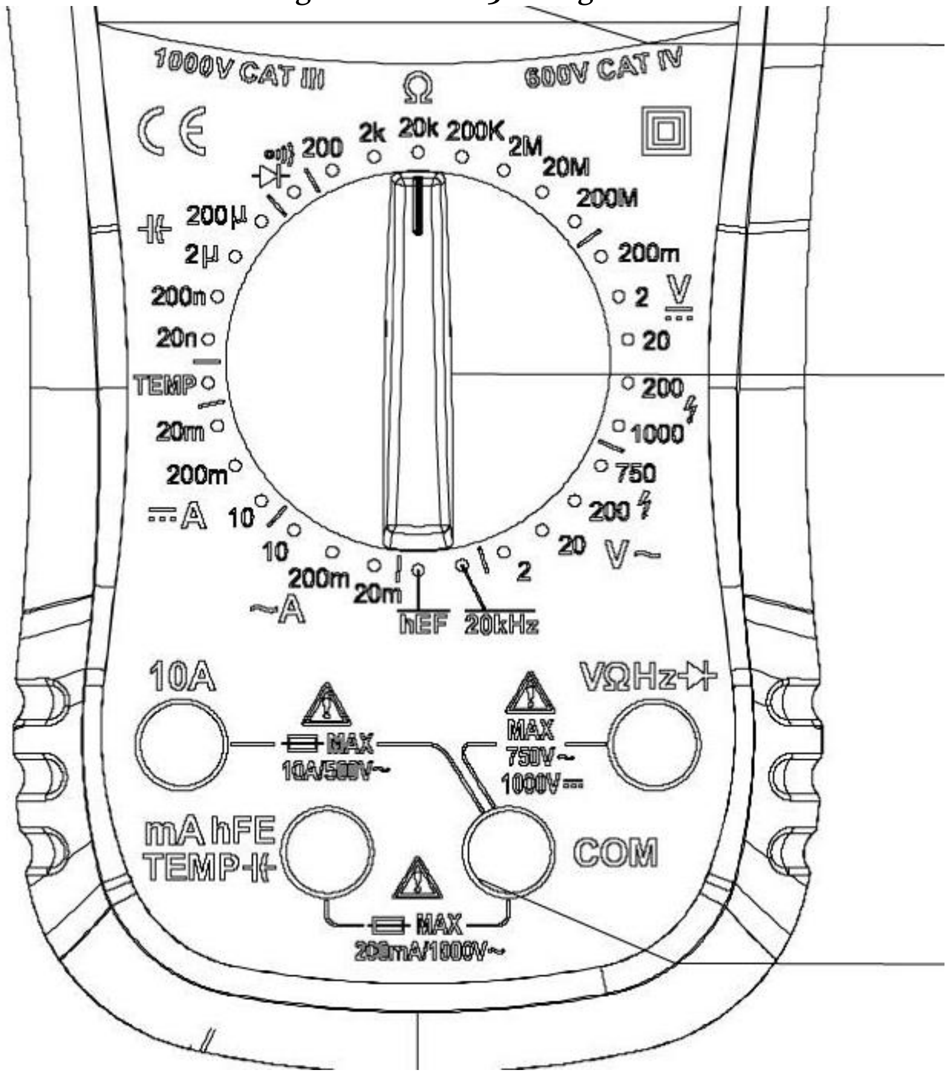
E. Taking Measurements Using a Multi-meter

A multi-meter is a tool that is used by technicians and engineers. It is a combination of several tools and is used to measure electric current, voltage, and resistance typically over several ranges of values. Some multi-meters also measure temperature.

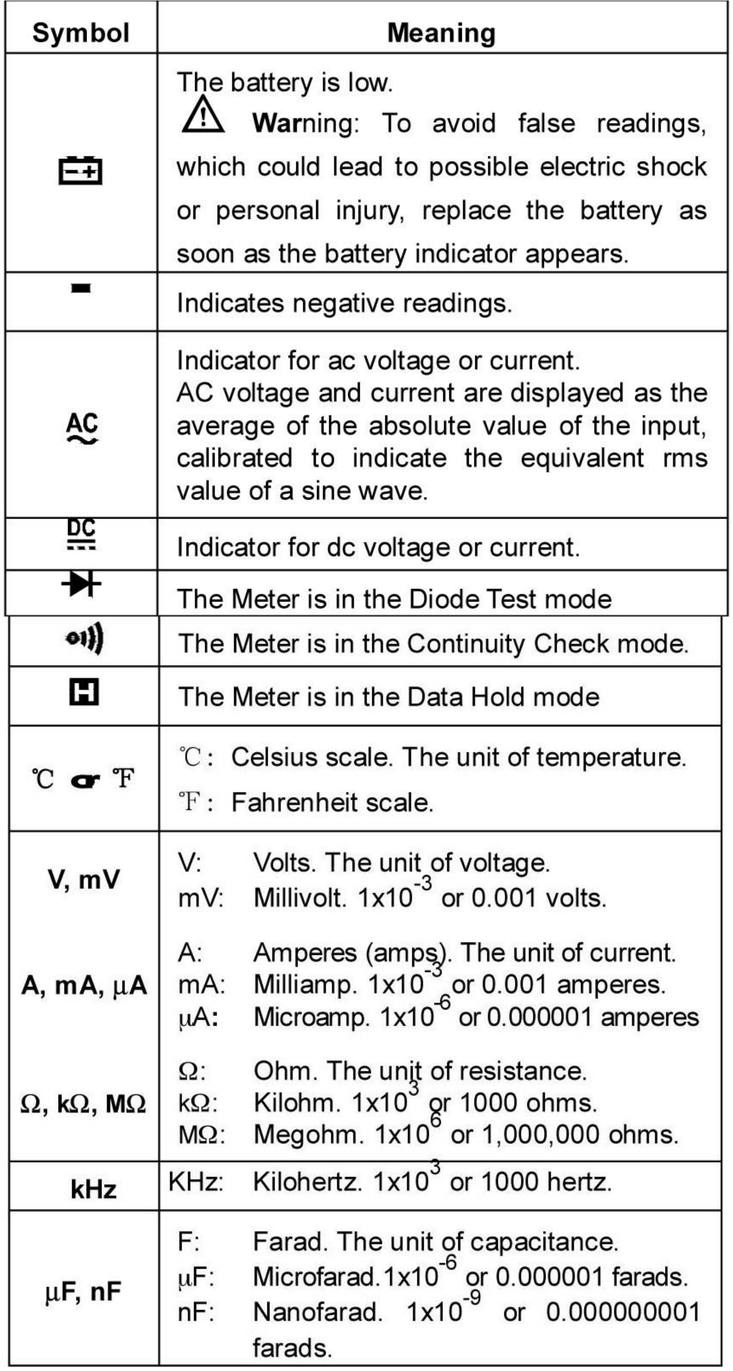
1) Meter Overview

The Global Specialties' Pro-50A Digital Multi-meter is recommended for use with this course. This is an excellent tool which can provide many years of use for students, technicians, and engineers.

The Dial Settings on the Pro-50A Digital Multimeter:

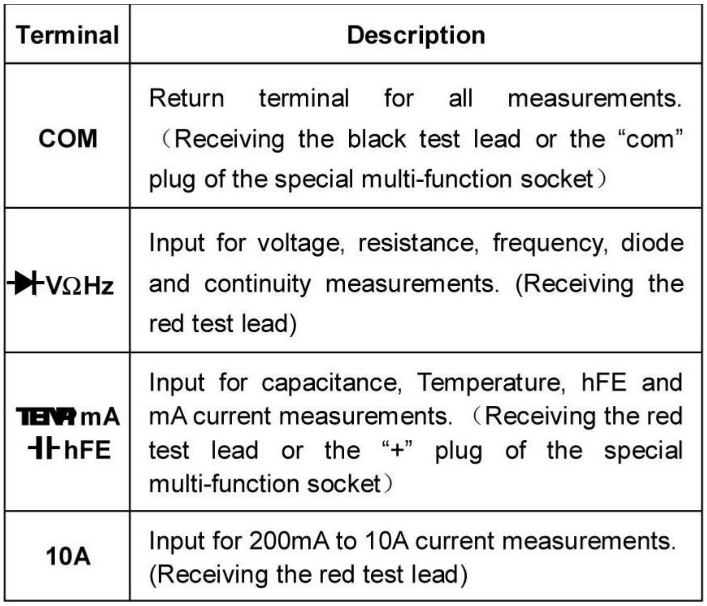


Symbols used in the display:



Symbol	Meaning
[-+]	The battery is low. ⚠ **War**ning: To avoid false readings, which could lead to possible electric shock or personal injury, replace the battery as soon as the battery indicator appears.
▬	Indicates negative readings.
AC ~	Indicator for ac voltage or current. AC voltage and current are displayed as the average of the absolute value of the input, calibrated to indicate the equivalent rms value of a sine wave.
DC ⎓	Indicator for dc voltage or current.
▶|	The Meter is in the Diode Test mode
•)))	The Meter is in the Continuity Check mode.
H	The Meter is in the Data Hold mode
℃ **or** ℉	℃: Celsius scale. The unit of temperature. ℉: Fahrenheit scale.
V, mV	V: Volts. The unit of voltage. mV: Millivolt. 1x10^{-3} or 0.001 volts.
A, mA, μA	A: Amperes (amps). The unit of current. mA: Milliamp. 1x10^{-3} or 0.001 amperes. μA: Microamp. 1x10^{-6} or 0.000001 amperes
Ω, kΩ, MΩ	Ω: Ohm. The unit of resistance. kΩ: Kilohm. 1x10^{3} or 1000 ohms. MΩ: Megohm. 1x10^{6} or 1,000,000 ohms.
kHz	KHz: Kilohertz. 1x10^{3} or 1000 hertz.
μF, nF	F: Farad. The unit of capacitance. μF: Microfarad.1x10^{-6} or 0.000001 farads. nF: Nanofarad. 1x10^{-9} or 0.000000001 farads.

Symbols used in the Terminals – where the black and red test leads are plugged in:



Terminal	Description
COM	Return terminal for all measurements. （Receiving the black test lead or the “com” plug of the special multi-function socket）
▶|VΩHz	Input for voltage, resistance, frequency, diode and continuity measurements. (Receiving the red test lead)
TEMP mA ⊣⊢ hFE	Input for capacitance, Temperature, hFE and mA current measurements. （Receiving the red test lead or the “+” plug of the special multi-function socket）
10A	Input for 200mA to 10A current measurements. (Receiving the red test lead)

Test Probes/Leads:
There is a bare metal area like a needle at the end of both the red and black test probes. These bare metal areas are the active areas of the probes and are used to come in contact with the device under test. When we refer to placing the red and black test probe onto a device we are actually referring to the bare metal pointed areas of the probes and not the insulated plastic areas where you can safely place your hands.

2) Measuring Resistance

Resistors are a common electronic devices used in many hobby robots and kits. We can use the Pro-50 Digital multi-meter to determine the value of a resistor.
First set up the multi-meter. Our first decision is the type of reading we are going to take, which in this case is Resistance. So we look at the dial and choose the appropriate scale for resistance. This is based on the measurement value for resistance which is what? Ohms. All resistance is measured in ohms.

Next we must select a range of values that we anticipate. What is the value of the resistor we will test? We identify the value of resistors by the color bands. The colors on our example resistor are: Brown, Black, Red, and Gold. We look those up using the color chart in the book and determine this is a 1,000 ohm or 1k ohm resistor. We choose a setting that is higher than what we anticipate, so we choose the 2k setting.

Next we put the black lead test probe area to one end of the resistor and the red test probe area to the other end of the resistor.

Now we look at the value displayed. An example value in the display now is .970 k which means that the value is 970 ohms. This is within the +/- tolerance of 5% for the resistor.

3) Measuring Voltage

One of the most common electrical measurements is Voltage. When dealing with hobby robots and other electronic kits, we are often working with batteries. If the batteries are low, the robot or kit may not perform as expected. So occasionally, we may need to check the voltage output from the batteries.

We can use a multi-meter to read the voltage and verify the amount of charge remaining on a battery.

We know that batteries are DC voltage, so we select that scale on the range and function selector dial. Next we need to know approximately how much voltage is to be read so we can set the meter. There are many types of batteries from very small to very large. A group of batteries may provide a wide range of voltages depending upon the type and configuration. Most batteries will have the amount of voltage printed on the side.

We can use the multi-meter to determine the voltage level of several different batteries. For example, we can test a AA battery which is 1.2 Volts. We set the range and function selector dial to 2 VDC (which stand for volts DC) range.

Place a red test probe onto the positive side and a black test probe onto the negative side of the battery. Read the display. If the battery is fully charged, it should be around 1.2, but if the battery is not charged, we may see a reading of .25 to 1.0.

F. Safety

Safety is a vital concern for all technicians and engineers. There are several topics involved in safety. In a lab environment, avoiding personal injury is number one, avoiding damage to components, and avoiding damage to tools and equipment (such as multi-meters) is also important. When working in any lab environment it is important to follow all posted safety precautions and to wear appropriate protective gear. Also participate in drills and know what to do in case of fire or an explosion. Here are some other hazards that can occur.

1) Eye Damage Hazards

When working with robots, safety glasses are important to protect your eyes. Hazards include (a) debris caused when using tools such as cutting, drilling, or soldering and (b) parts or objects that may fly up unexpectedly when the robot is operating.

2) Shock Hazards

When working with electronics, one common hazard is electrical shock. With our hobby robot the battery voltages we use are too low to cause shock hazards. However, when the multi-meter is used to test AC devices, those devices can cause shock hazards.

3) **Burn Hazards**

There is an opportunity to suffer burns through misconnection or making errors when wiring, causing an overheating of wires or components. As an example, miss wiring can cause batteries to overheat. Use caution when touching the battery case. If you smell something burning, use a pencil, pliers or other device to remove the batteries.

Another burn hazard is with the use of the soldering iron. The tip of the soldering iron is very hot and will easily cause burns on the skin if touched. Also, melted solder may cause burns if dripped onto skin or touched before it has time to cool. Follow safe soldering procedures to avoid burns.

4) **Component Hazards**

Electrostatic damage (ESD) may occur when walking over carpet and then touching an electronic part. It is important to use a grounding wrist strap or touch a grounded device (radiator, metal water pipe, computer case, etc.) before touching ICs and sensitive electronic devices.

Another way to damage equipment is to miss-wire it - for example reversing the power supply leads to the device. Carefully follow the schematic drawings and all instructions to ensure that the positive and negative wires are placed properly to avoid this. Also use red and black wires as indicated. Negative (Black wire) connections are indicated as GND or VSS or V- on schematics. Positive (Red wire) connections are indicated as VCC, or VDD or V+.

It is also possible to damage the electronic components by overheating. Care must be taken when soldering to not overheat the components.

5) **Instrument Damage Hazards**

Taking electrical measurements using instruments is a common task for technicians and engineers. It is important to avoid damaging instruments as they can be very sensitive to faulty operations. Most commonly, damage occurs when the wrong function or scale is selected when testing. This is especially true of multi-meters and oscilloscopes. It is important to learn the proper operation of all equipment before use.

Any unfamiliar lab equipment, tool or instrument can pose a hazard. When you don’t know how a device works, ask a lab technician, teacher or supervisor to demonstrate safe operation.

These are just a few of the most common safety hazards when working with electronic devices. Always use common sense take precautions when working with electricity.

6. Prototyping with Breadboards

A prototype is an operating version of an engineering solution. It is often made with different materials (cheaper and easier to work with) than the final version. Prototypes allow you to test how a solution will work and possibly even show the solution to users for feedback.

Engineers and technicians create prototypes using readily available materials, breadboards, construction kits, storyboards, or other techniques that help to create a solution quickly and with little cost. These are understood to be mockups of a final solution, not the real thing.

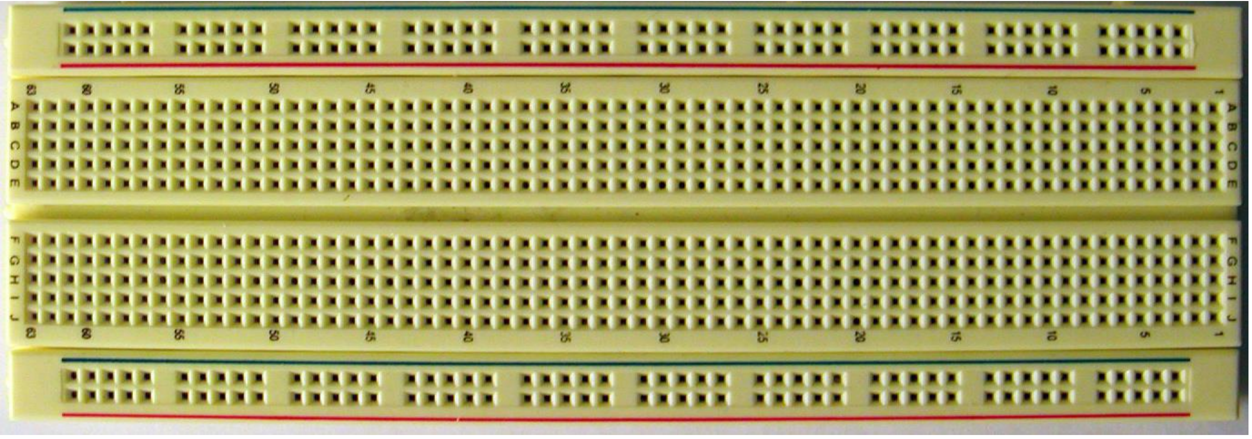
A. Engineering Prototype Process

A prototype is a temporary device built to test an idea. It is not usually designed to last very long but simply as a test. An electronics engineer or technician will often build a prototype of an electronic circuit using a solderless breadboard. The breadboard makes it easy to try out connections of components to test the operation of a circuit to perform some function, or solve a problem. It is also used to test component to make sure they work.

A breadboard is used because it is easy to insert parts and take them out and move them if necessary. It also provides a grid pattern that makes it easier to keep track of components that are connected together.

B. Using a Bread Board

To build a prototype circuit we will use a device called a solder-less breadboard. This device shown here has some unique features.



First notice that the breadboard has a large number of holes. Each hole is designed to accept the insert of a wire or the end of an electronic device.

The columns have numbers on the side which can be used to reference each one. The columns are in groups of five, 1, 5, 10, 15…65. The rows have letters – ABCDE and FGHIJ.
The holes are connected on the back of the device such that all holes in a numbered column, are connected together. This means that any wires or devices that are plugged into the same row are connected.

Two special areas of holes at the top and bottom of the breadboard are usually used for connecting power. On the top a solid red line below a series of holes indicates that all these holes are interconnected. Typically we would connect positive power to this row by inserting a red wire anywhere along these holes.

In a similar way a solid blue line appears above a series of holes indicates that all these holes are interconnected. Typically we would connect negative or ground power to this row by inserting a black wire anywhere along these holes and then connecting that wire to power negative or ground.

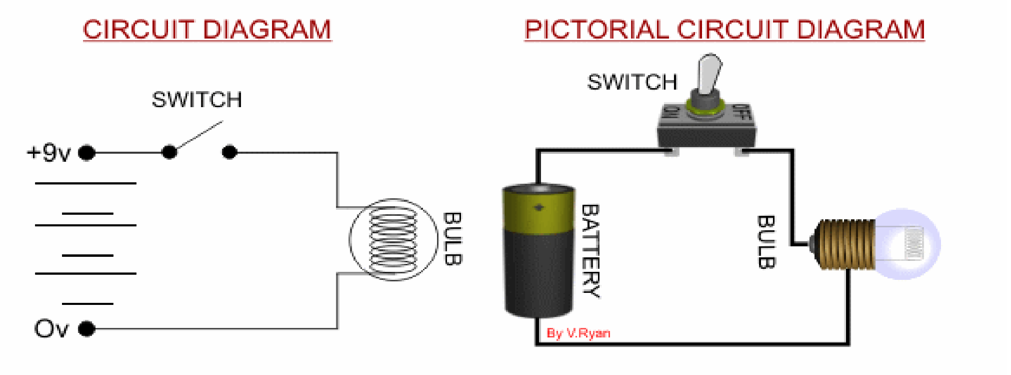
At the middle of the board there is a separation. There is no interconnection between the areas above and below this gutter. Items in rows ABCDE are not connected to items in rows FGHIJ, and items in the power red and blue areas at the top are not connected to items in the power red and blue areas at the bottom.

C. Activity - Our First Circuit

To learn how to use the bread board, we will build a simple circuit. In electronics, a *circuit* is a path between two or more points along which an electrical current can be carried. When we create a circuit, it consists of a group of devices and wires connected in such a way to create a specific result or purpose. To put it another way, a circuit is a group of things wired together to create something useful.

What are the minimal things we need to create an electrical circuit? First we need a power supply: or a method of creating electrical energy.

Next, we need wire or conductors. Wires or conductors are used to allow electrons to move throughout our circuit. They are like pipes in a plumbing system. The force that causes electrons to move in a common direction through the circuit is called *a difference of potential* or voltage.



Next, we need all the other devices required to make our circuit. In this example we have a light bulb and a socket to hold it, a battery, and a switch. Look at the two drawings. It is easier to understand what is going on from the pictorial drawing on the right.

Notice we have a battery as a power source. In the schematic or circuit diagram on the left, we use a symbol for a battery. Next we have a line leading from the top of the battery to the switch. Lines like this represent a wire connecting two or more devices. On the left there is a symbol for a switch. On the right we have a picture of a switch. Another line leaves the right of the switch and goes to the light bulb. This represents another wire. On the left the light bulb is shown as a symbol.

Finally another wire, (shown as a line) goes from the light bulb to the power supply. This completes the circuit and shows the path that electrical energy flows from the negative side of the battery to the positive side of the battery.

Our circuit has a switch. A switch is an electrical device that makes a connection when it is activated. We are all familiar with switches as we use them every day to turn things on and off. There are a huge number of switches of all types.

How does this circuit work? The battery is our source of electrical energy or current. We connect the battery to a switch that controls whether or not the battery is connected to a light bulb. If the switch is activated, electrical energy can flow from the battery to the light bulb, and the light bulb glows. If the switch is off or not activated, then no electrical energy can flow, and the light bulb does not glow.

1) Prepare the parts

Now it is time to build the circuit. First gather these parts:

- The breadboard
- The circuit drawing or schematic
- A switch
- A bulb
- A battery
- Short pieces of black and red wires

2) Build the Circuit

Watch the video(s) for this activity. Follow the steps in the video and build the circuit on the breadboard.

After building the circuit, use the multi-meter to test the amount of voltage and current going thru the circuit. Use the worksheet for this activity to record results.

- Reading Voltage
- Reading Current
- Reading Temperature
- Recording Readings

7. Soldering PC Boards

Solderless breadboards are used to make prototypes and to test circuits, but when we want a circuit to last for more than a few days, we solder the components together onto a printed circuit board.

Soldering is a process in which two or more metal items are joined together by melting and then flowing a filler metal into the joint—the filler metal having a relatively low melting point. Soldering is used to form a permanent connection between electronic components. The metal to be soldered is heated with a *soldering iron* and then *solder* is melted into the connection. Only the solder melts, not the parts that are being soldered. So solder is like a metallic "glue" that holds the parts together and forms a connection that allows electrical current to flow.

A. Printed Circuit Boards

Printed Circuit Boards (PCBs) provide a highly reliable method of creating electronic devices at high volume with lower costs. When creating a prototype with a solderless breadboard, we gain experience in using wires and building circuits one device at a time.
PCBs eliminate the need to build circuits one wire at a time because the wiring is all located on a board surface substrate.

First we will describe a through-hole mounted PCB. Many circuit boards used in kits for students and hobbyists use this type of PCB. The name is derived in that the components are mounted with their leads or wires pushed through from the component side of the board to the circuit side.

1) Thru-Hole PCB

PCBs are generally built by using a fiberglass board which is made up of multiple layers of fiberglass held together with epoxy (a special type of glue). The fiberglass boards are produced by coating fiberglass with an epoxy resin then placing that into a laminating press with a very thin sheet of copper on its surface. The copper is treated with a chemical that helps it adhere or bond to the fiberglass. The fiberglass sheet and copper sheet are placed into a press and heat and pressure are applied that causes the epoxy to cure and the copper becomes bonded to the fiberglass surface.

Next a negative image is made of the desired circuit where all the copper that does not represent a wire is removed. This is the printing part of a printed circuit board. Next the image is transferred to the copper surface with a photographic process. This process creates a photo resistive layer that causes the places where wires are located to become hardened so they are not removed in the next step. Next an etching solution is used which eats away the copper except where the wires are located.

On the opposite side of the PCB, a silk screening process will print the identification and location of various components to be mounted on the board.

This is called the component side of the PCB. Next holes are drilled through the PCB where each component is to be mounted with wires that go through the holes. The wires are soldered to areas called pads on the circuit side of the board, providing connection to the copper wire areas - which are called traces. The amount of current that a trace can carry is determined by the width of the trace and the thickness of the copper. Tables are available that give the trace width and copper thickness for various current carrying capacities.

PCBs are made with various thicknesses of copper rated in ounces per sq ft. Fiberglass is used for the board portion of a PCB for several reasons. 1) it is a good insulator 2) it has high strength 3) it has good dimensional stability meaning it doesn't shift excessively in size over the temperature range encountered during manufacture and by most electronic devices.

2) Surface Mount PCBs

Another type of PCB which is most often used in high-volume electronic devices is a surface mount PCB. This type of PCB has a majority of its components mounted directly to the surface and has leads which are directly connected to traces on the mounting side. In other words this eliminates the need for drilling holes. Surface mount devices are also smaller in size than those with wires attached. A surface mount resistor or capacitor can be much smaller than one designed for thru-hole mounting. It is also much easier to use robots and automated machines for placing surface mount devices on the PCB.

Because components are smaller and can be placed closer together, the size of the PCB becomes much smaller. The manufacturers are making the components increasingly smaller. With reduced size and using automation, boards can be produced at a lower cost. With the lower cost, most surface mount boards are replaced instead of repaired.

Surface Mount technology is designed primarily for automated assembly. However it is possible with care, skill, and training to manually assemble some surface mount PCBs. the soldering techniques which are used are more complex and require magnification to assist in the process.

3) Limitations on PCBs

Routing the traces on a PCB can become quite complex. because the copper is all on a single layer, it is necessary to route the traces in such a manner that they do not touch or short out between traces. So components must be located on the board in such a way to avoid crossing traces when interconnecting components. At one time this was all done manually by an engineer who specialized in PCB layout. Today there are software programs that will do a good job of routing traces to avoid conflicts crossing and shorting of traces.

4) Flexible Printed Circuit Boards

Another type of circuit board is the flexible circuit board. They are made from materials which are much thinner, but still rugged. They can be used as an interconnect between two other circuit boards or as a connection to a display, keypad, camera, or other device. They are a cost effective solution for connecting devices that require multiple conductors, and may be used instead of a cable to avoid the connectors that are required at the end of cables.

The reason for making it flexible is that it often needs to bend at the connecting point or to go around some object within a case. But once connected, they often do not have to bend much. They are also used in wearable technology, where they need to bend with clothing. To make it more rugged, sometimes flexible circuit boards are encased inside of clear plastic with laminate or with plastic in a process call potting.

B. Soldering Process

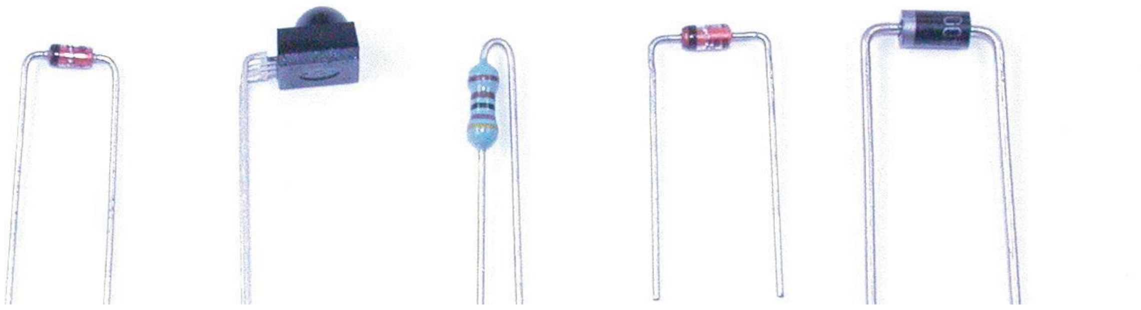
While soldering small parts you will need some tricks and tips to reach the soldering area. A tool called a "third hand" can solve the problem of holding onto parts and holding onto the PCB board, freeing both hands for the soldering process.

Some parts (transistors, LEDs, ICs, switches, capacitors and jumper) are already prepared to fit into the PCB (printed circuit board), but diodes and resistors have to be prepared to do so.

Resistors are to be placed vertically in the ARX ASURO PCB. To do so, one leg remains in its original position, the other one is to be bent 180 degrees. Bending must be done at a curvature of 2.5 mm diameter at a distance of a few millimeters from the resistor body to avoid damage to the part.

Diodes are inserted in horizontal position. To do so both legs have to be bent (eg. by pincers) at a distance to fit into the holes of the PCB.

Electronic parts with legs bent prior to soldering:



Overview of Soldering Process:

A. Part

B. Leg, solder pad, and bending area have to be heated simultaneously

C. Solder must flow into the drilling hole

D. Solder

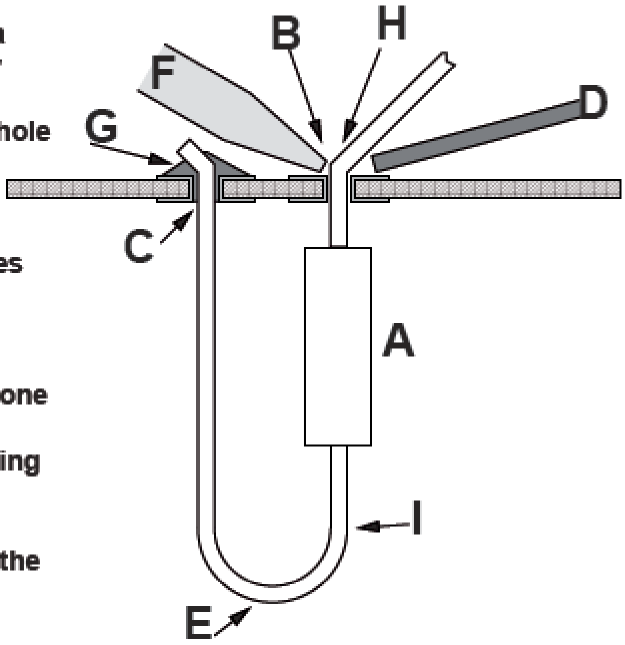
E. Round bending area without edges

F. Soldering tip

G. Neat, vulcano-shaped soldering cone

H. Bending legs to prevent parts falling out from the PCB

I. Bend area at some distance from the part



1) Safety Precautions

- Caution: A soldering iron can heat to around 400°C, which can burn you or start a fire, so use it carefully.
- Unplug the iron when it is not in use.
- Keep the power cord away from spots where it can be tripped over.
- Take great care to avoid touching the tip of the soldering iron on a power line. If a power cord is touched by a hot iron, there is a serious risk of burns and electric shock.
- Always return the soldering iron to its stand when it is not in use.
- Never put the soldering iron down on your work bench, even for a moment!
- Work in a well-ventilated area.
- The smoke that will form as you melt solder is mostly from the flux and can be quite irritating. Avoid breathing it by keeping your head to the side of, not above, your work.
- Some solder contains lead, which is a poisonous metal. Wash your hands after using solder.

2) Preparing the Soldering Iron: Tinning the Tip

- Place the soldering iron in its stand and plug it in.
- Wait for the soldering iron to heat up.
- Moisten the sponge.
- Wipe the tip of the iron on the damp sponge. This will clean the tip.

- Melt a little solder on the tip of the iron.
    - This is called tinning and it will help the heat flow from the iron's tip to the joint.
    - The solder should flow onto the tip, producing a bright shiny surface.
    - If the solder will not flow onto the tip, clean it by wiping it on the wet sponge.
    - When tinned, wipe excess solder off on the wet sponge.
    - You do not need to tin the tip before every joint, but you should re-tin it if it has gone dull when the soldering iron has not been used for a few minutes.
    - Check the manufacturer's instructions related to tinning the tip.
- The tip of the soldering iron should be a shiny silver color. If it is black and pitted, replace it with a new one.

3) Soldering

Solder needs a clean surface on which to adhere.

- Buff the copper foil of a PC board with steel wool before soldering.
- Remove any oil, paint, wax, etc. with a solvent, steel wool, or fine sandpaper.

To solder, heat the connection with the tip of the soldering iron for a few seconds, then apply the solder.

- Heat the connection, not the solder. The solder should melt from the heat of the parts and flow around the part.
- Hold the soldering iron like a pen, near the base of the handle.
- Both parts that are being soldered have to be hot to form a good connection.

4) Troubleshooting Soldering

Solder will not flow.

- The parts to be joined may be dirty. Remove the solder and clean the parts.

The connection looks grainy or crystalline.

- Parts were moved before the solder was allowed to cool.
- Reheat to form a good joint. You may need a larger soldering iron to heat connections adequately.

The tip is oxidized.

- Soldering is much easier with a shiny, clean tip.
- Clean the tip with a damp synthetic sponge while the iron is hot.
- To avoid oxidizing the tip, do not leave the iron plugged in when not in use.
- Do not use the iron at a higher temperature than is necessary to melt solder.

- Clean the tip of the iron on a damp synthetic sponge as soon as it starts to change from a silver color.

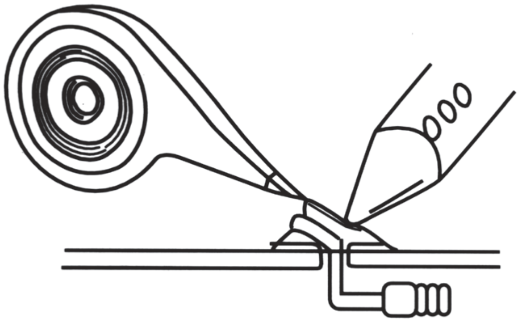
There is too much or too little solder.

- Using too much solder can cause a solder bridge, which means that two adjacent joints are accidentally connected.
- Using too little solder might result in poor electrical continuity between the board and component. The connection should be smooth, shiny, and rigid.

C. De-Soldering Process

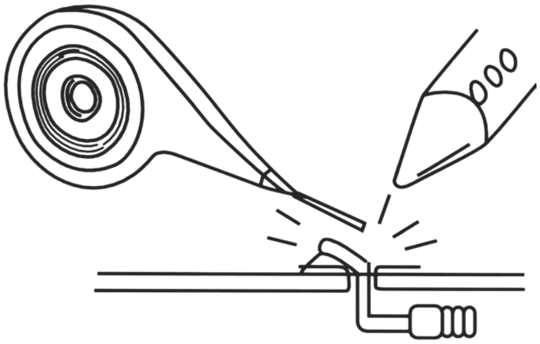
When, occasionally, a part has landed on a spot where it does not belong then you have to remove it. The ARX ASURO has a double sided PCB with trough metalized holes, this makes the removal of the parts not very easy.

- Add some flux to the soldering of the part which must be removed (a simple way of doing this is to add some extra solder).
- When all the solderings of the part are heated try to remove the part with pliers from the PCB. At the end you can easily remove the solder with de-soldering wick (or braid).
- Put the soldering wick on the solder(pad) [see, step 1 & 2, below]. You may also do this when the pad is still in the PCB! Heat the wick and the solder together.
- At some point the wick will suck the solder into the copper braid. At this point, REMOVE the soldering iron and wick quickly.
- You may repeat this on the other side of the hole when there is still solder in it.



Step 1

Put the copper braid over the solder(pad) of the component which must be removed. Heat the braid and the solder(pad), the braid will now suck the solder.



Step 2

Remove the soldering iron and copper braid at the same time.

D. Learning Soldering Activity

To build up skills in soldering before working with the ARX ASURO, we will use a small PCB kit with a few parts.

- Follow the instructions in the activity video to see how to solder the parts to the board.
- Follow the soldering process listed above to solder the parts.

SECTION 2 Prototypes with Breadboard

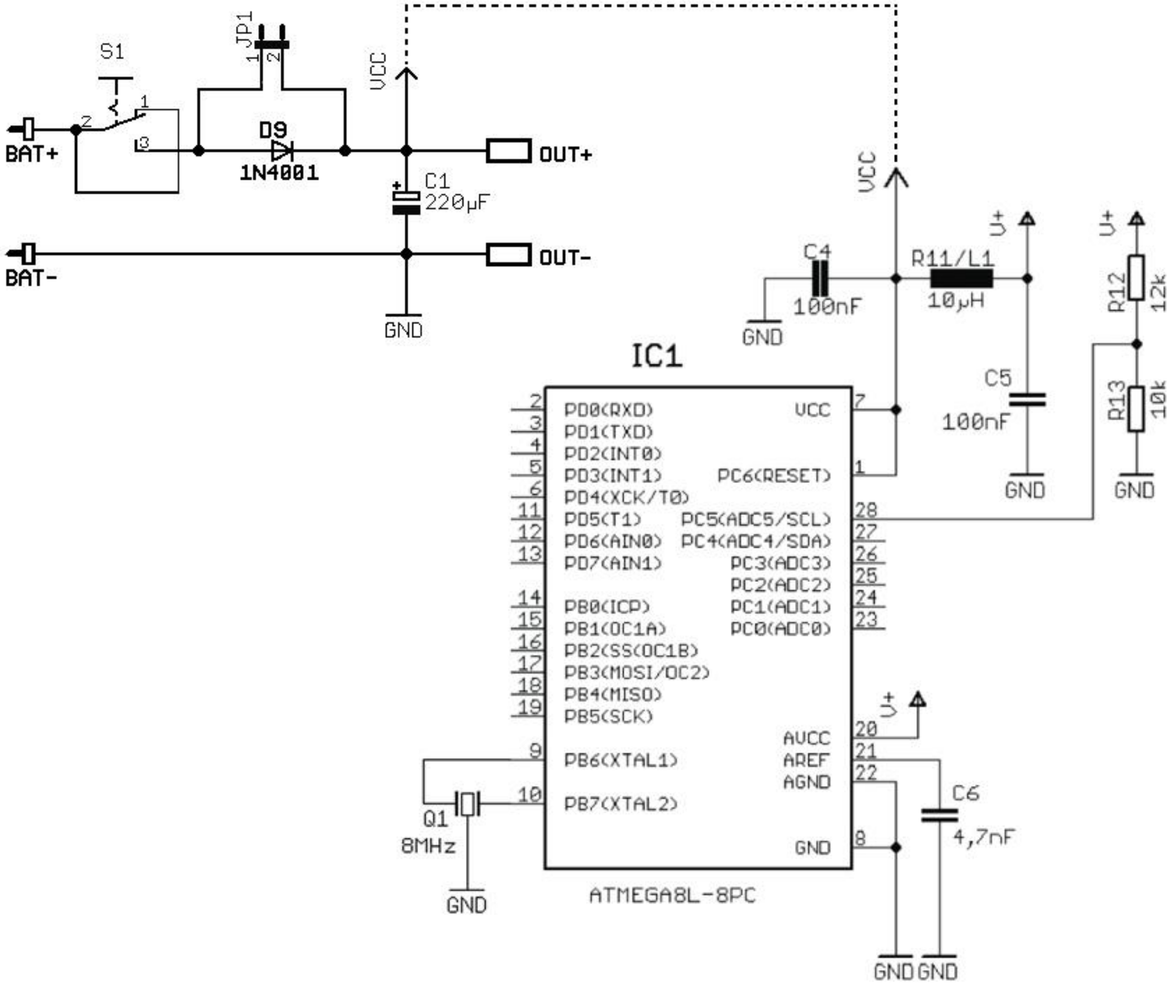
- *Power Supply Circuit*
- *Status Circuit*
- *Collision Detection Circuit*
- *Motor Control Circuit*
- *Encoder Circuit*
- *Line Follower Circuit*
- *Microcontroller Circuit*

8. Power Supply Circuit

All electrical and electronic devices require power. The two most common power sources are batteries or DC for portable devices and 115V AC (Volt AC) for residential and commercial building power.

If the device has a microcontroller the most typical voltage will range from 3 to 5 Volts DC. It is very common to supply power to a microcontroller with a device called a wall wart. This is a device that plugs into a standard 115V AC outlet and provides an output of DC voltage. If you have a cell phone or a laptop computer you probably have one of these devices to charge the battery.

Other electrical devices (such as desktop computers) have 115VAC input and then internally convert to DC and provide DC voltage for control circuits. Typically these types of supplies have a transformer that has 115vc input and a low voltage AC output. This low voltage AC output is then converted to DC power.



A. ARX Power Supply Circuit Components

The ARX power supply is a relatively simple device consisting of a battery holder that contains four batteries wired in series which provides approximately 6 volts DC power, and a switch for turning the power on and off. There is also a power supply circuit containing components which assure that the power supply voltage level is adequate for operation. A voltage monitoring function uses an analog to digital converter to determine the level of voltage.

1) Chargeable vs. Rechargeable Batteries

The ARX power supply uses four series connected AAA batteries. If non-rechargeable batteries are used, the batteries will yield approx. 6 volts. If rechargeable batteries are used, they will yield approx. 5 volts when charged. A Diode (D9) provides a voltage drop that reduces the 6 volt non-rechargeable battery configuration down to approx. 5 volts to protect voltage sensitive devices. The Diode (1N4001) provides a voltage drop of approximately .6 to .8 volts.

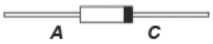
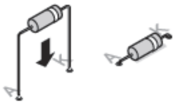
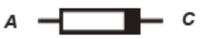
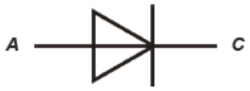
IMPORTANT: *If rechargeable batteries are used it is necessary to provide a jumper bypassing Diode D9 to allow 5 volts of power to be provided to the ARX. This also means we use either chargeable or rechargeable batteries, and not switch back and forth with the ARX ASURO*.

2) Diodes

The main property for a diode is its ability to pass a current in only one direction from its anode side to its cathode side. It is very important that the diode be inserted properly. There is a band on one end of the body which indicates the Cathode or negative side. The schematic symbol indicates the direction of current flow from anode to cathode by a symbolic arrow (see Circuit symbol in table below).

Examples of Diodes

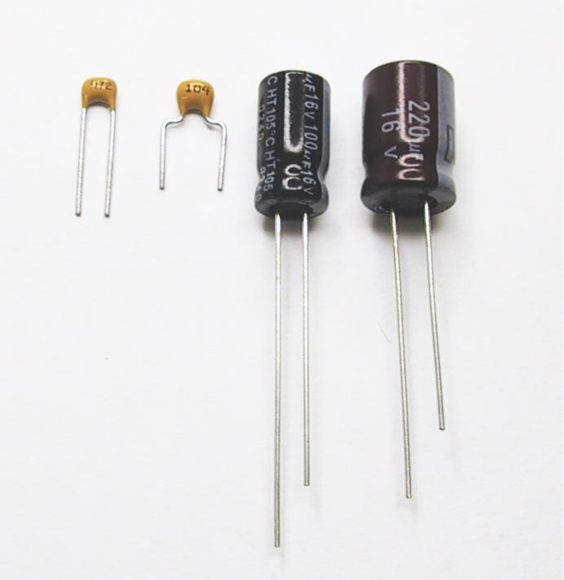
The diode we are using in this circuit is 1N4001, but it is similar to the description of the Zener diode shown in the back of the ARX ASURO manual. Here we see the schematic symbol for the diode and note that the legs are bent 90 degrees before inserting the diode into the breadboard or PCB board.

Component	Assembly	PCB symbol	Circuit symbol
(Zener)Diode  Read the imprints of the parts and take care not to interchange the zenerdiodes with 1N4148 diodes!	 Notice correct polarity!	 	

3) Capacitor

Another device used in the power circuit is a capacitor. ***Capacitance*** *is the ability of a device to store electrical energy in an electrostatic field. A* ***capacitor*** *is a device that possesses a specific amount of capacitance.* The amount of energy stored in a capacitor is in proportion to its size.

When a DC voltage power source is connected to a capacitor, current flows until the capacitor is charged, and then the current stops flowing. The capacitor's voltage is equal to the voltage of the power source (in our case 5 or 6 volts). The capacitor used in the power circuit helps to stabilize the power in the circuit.

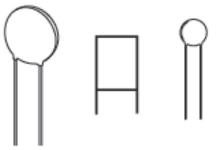
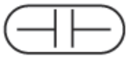
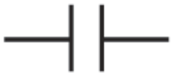


Examples of Capacitors

The basic unit of capacitance is the farad (F). A farad is the amount of capacitance that can store 1 coulomb(C) of charge when the capacitor is charged to 1 volt. The farad is too large for ordinary electronic use, so the microfarad (µF) and picofarad (pF) are typical measurements for capacitors. One microfarad is one millionth of a farad and one picofarad is one trillionth of a farad. The capacitor we are using is 220 µF, (which is identified with the numbers 220 printed on the side).

As shown in the table below (from the back of the ARX ASURO manual) we see the Circuit symbol for a capacitor. Capacitors come in many types and styles. An Electrolytic capacitor is used here because it offers large capacitance for small size and weight. Also note that this capacitor has radial leads and that means the leads do not have to be bent before inserting them into the breadboard.
An imprint and/or a notch ring at one side will mark polarized capacitors. The polarity mark is normally a large minus ("-").

Capacitors are available in a great variety of application forms dependent on maximum allowable voltage, capacitor value, suitability for DC or AC currents, capacitive tolerance, etc. Capacitor values will often be imprinted on the capacitor. Larger capacitors will be marked by full text labels. Smaller capacitors usually are labeled by a three-digit code, in which the leading part of the capacitor value is coded in both leading digits and the trailing digit codes the number of zeroes in the capacitor Picofarad-value. An imprinted number 472 for example means 4700 pF, which may also be written as 4.7nF.

Component	Assembly	PCB symbol	Circuit symbol
Ceramic capacitor 			

Safety Precaution: *It is important to note that because a capacitor stores energy, it should be treated as though it is charged. Never touch both leads of a capacitor in a circuit with your hand because you could get shocked. Discharge a capacitor by shorting both leads together. A capacitor in a circuit can hold a potential energy indefinitely if it does not have a discharge path.*

4) Calculations Used With Capacitors

Note these calculations are not required to assemble the prototype, but are included here only as part of learning about electronic devices.

Capacitors can be connected in series, in parallel, and series-parallel configurations. When capacitors of different values are connected in series, the smaller capacitors charge up to the highest voltage. The total capacitance of capacitors in series is calculated as:

$$\frac{1}{Ct} = \frac{1}{C1} + \frac{1}{C2} + \frac{1}{C3} \dots + \frac{1}{Cn}$$

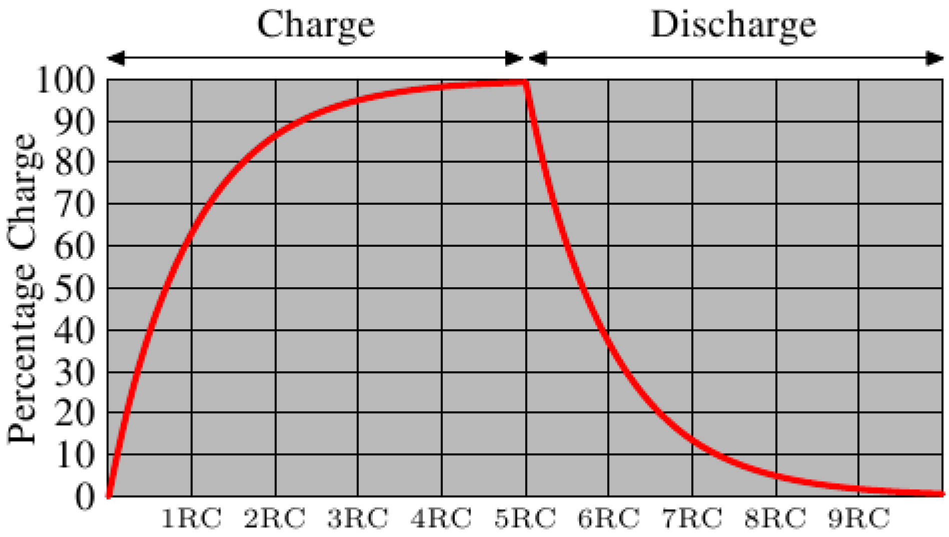
Capacitors connected in parallel adds the total capacitance area. This makes the total capacitance equal to the sum of the individual capacitances:

$$Ct = C1 + C2 + C3 \ldots Cn$$

The time it takes for a capacitor to charge and discharge is directly proportional to the amount of resistance and capacitance in the circuit. The time constant is expressed as:

t = RC

where t= time in seconds, R= resistance in ohms, and C= capacitance in farads. ***Note that it takes approximately 5 time constants to fully charge or discharge a capacitor.*** The time for charge and discharge is not a linear progression, but is curved as shown here:



A non-polarized capacitor or a polarized capacitor is a storage component for electrical energy. Its energy status depends not only on momentary parameters, but on the preceding conditions as well. The voltage between its terminals depends on the electrical charge it has accumulated in the past. Assuming we take an initially empty capacitor with a capacity (for energy storage)

$$C[Farad, F = \frac{As}{V}]$$

and a constant charging current I, we will measure at its terminals a voltage *V* dependent on time t of

$$U = \frac{I}{C} * t$$

Just in case the charging current is varying in time and the capacitor voltage at the point of time t = 0 is U0, we may calculate the voltage at other points of time by the following integral formula:

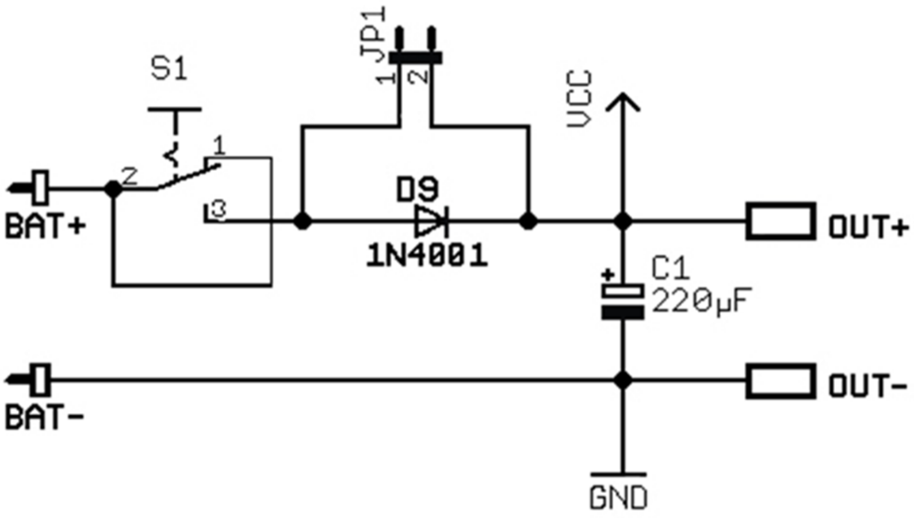
$$U(t) = \int_0^t \frac{I(T)}{C} dT + U_0$$

If an empty capacitor is connected by a series resistor to a constant voltage source U0, the capacitor is being charged to a voltage U(t) at a point of time t by:

$$U(t) = U_0(1 - e^{-\frac{t}{RC}})$$

B. Schematic Diagram for DC Power Circuit

The prototype DC Power Circuit we will build is represented in this modified schematic diagram.



C. Parts Needed for Prototype

For building the power supply prototype, we will focus only on one section of the circuit shown in the above diagram. Therefore, we will not use all the parts that make up the entire circuit. We will not use the switch or the jumper in our prototype.

From ARX ASURO Kit:

- Battery holder
- 4 (AAA) batteries
- Diode D9 (1N4001)
- Capacitor C1 (220 µf)
- Resistor (470 ohm), and Red LED

From Separate Kit:

- Breadboard
- Wire spools– Red and Black

Tools:

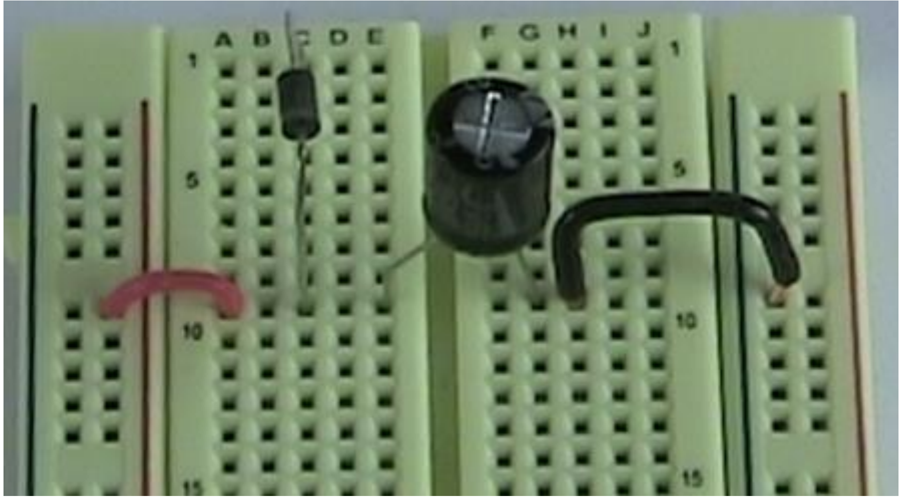
- Digital Multi-meter
- Pliers with wire stripping area

D. Power Supply Prototype Activity

Assemble and wire the prototype using the breadboard, the schematic diagram, and parts as illustrated in the Power Supply Prototype video.

Test the circuit following the instructions in the video and the activity.

This is an example of the completed prototype circuit.



9. Status Circuit

The status circuit is the simplest of the light related subsystems. The ARX ASURO robot uses LEDs as signals about the operation of various subsystems. When the LEDs are lit or flashing, or a certain color, it is an indication of the status of how the subsystem is functioning. The microcontroller programs monitor operation of the subsystems and turn on or off the status LEDS to communicate with the human operator that the system is operating properly or that something has gone wrong.

A. Status Circuit Components

The primary components which make up the status circuit are LEDs and resistors which protect the LEDs.

1) LEDs

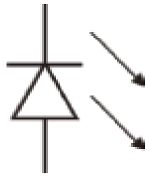
Light Emitting Diodes (LEDs) are the most common device used to indicate the proper or improper operation of electrical and electronic devices for the following reasons:

- They consume very little energy
- They last a very long time
- They are inexpensive
- They come in different colors
- They are low voltage devices
- They are simple to connect
- They can come in different colors in one device
- They are easy to control from microcontrollers
- In addition to being available in visible light they are also available in infrared and other non-visible lights.

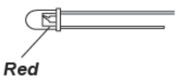
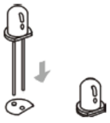
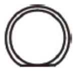
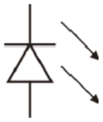
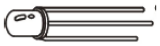
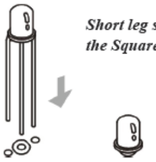
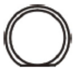
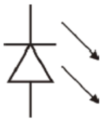
LEDs are diodes which emit light. The main property for a diode is its ability to pass a current in only one direction from anode to cathode. It is very important that the diode be inserted properly. On an LED, one leg is shorter which indicates the Cathode or negative side. The schematic symbol indicates the direction of current flow by a symbolic arrow.

2) LED Symbols

This symbol in a circuit diagram represents an LED. The two small arrows signify light being emitted. The large arrow indicates the direction of current flow.



The table below (part of table in appendix H of ARX ASURO manual) shows the four different types of LEDs used in the ARX ASURO. The first column (component) indicates how to identify the component. The second column (assembly) gives directions on how the LED is assembled or inserted. The third column (PCB symbol) shows the symbol used on the PCB board for the LED. And the final column (circuit symbol) shows the symbol used in a circuit diagram for this component.

Component	Assembly	PCB symbol	Circuit symbol
LED red D15-D16  	 C (Cathode) = Short leg should be on flat side!	 Flat side is K = -	
DUO LED D12 	 Short leg should be in the Square path		

3) Status LEDs on the ARX ASURO

So those are all logical and rational reasons for using LEDs so how are they used on the ARX ASURO robot? Looking at the schematic we find that one of the status LEDs is D11. When we look at the parts list, we see that D11 is a bicolor LED, red and green. The green portion of the LED is controlled by PIN 14 of the microcontroller through the current limiting 470 ohm resistor, R31. The red portion of the LED is controlled by Pin 4 of the microcontroller through the current limiting 470 ohm resistor, R10.

The logic for turning on the LEDs goes like this:
If only one of the LEDs is energized (by applying a positive signal to the current limiting resistors) then the LED will either glow Red or Green. We can make the LED glow red by applying a positive (1) to Pin 4, or make it glow Green by applying a positive (1) to Pin 14.

If we apply a positive (1) to both Pin 4 and Pin 14, the LED will glow orange. This is how we can achieve more than two colors from a bicolored LED. In fact, we can have even more colors by rapidly switching off the two pins 4 and 14 or by adjusting the relative values of the current limiting resistors (using different resistors). We will cover more about this in the programming section.

4) Back Left and Right Status LEDs

D15 and D16 are the Left and Right Back Status LEDS. The control circuit for these LEDs is more complex. This is an instance where we use more than one Microcontroller pin to control an on/off device. If Pin 13 is negative (0) then both D15 and D16 will glow dimly if Pin 23 and Pin 24 are both low (0). However if Pin 23 is High (1) and Pin 24 is low and Pin 13 is Low (0), then D16 will glow brightly. Conversely if Pin 23 is Low (0) and Pin24 is High (1) and Pin 13 is Low (0) th3n D15 will glow brightly.

5) Encoder IR LEDs

Two IR LEDs (D13 and D14) are used for the Encoder circuit. Both LEDs are controlled, turned on, by applying a high (1) to Pin 13 of the microcontroller. Current is limited by the 470 ohm resistor, R22. We will cover more details about the encoder circuit later, and we will cover how the encoder circuit is controlled in the programming section.

6) LED Operation

LEDs (or Light Emitting Diodes) pass current in only one direction from anode to cathode. The LED will start illuminating as soon as current flows in a forward direction. The polarity of the LED is marked by a difference in the length of the legs. The shorter leg and flat side of the LED is the cathode. We pay close attention to how the LEDs are inserted to ensure that the current flows in the proper direction.

7) Four Status LEDS

There are LEDs in the front and in the back of the ARX robot. The status LEDs are:

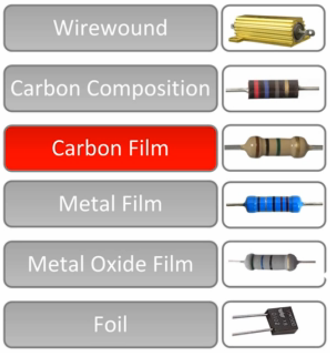
- Top-LED (D12) green /red
- Back-LED (D15) left
- Back-LED (D16) right
- Front-LED (D11) on the bottom side of the ARX ASURO (Line Tracing status)

These four LEDs (D11, D12, D15, D16) are used in various combinations to indicate status and proper operation of the ARX systems. They are also used during troubleshooting to isolate potential error sources. The troubleshooting procedures using the status LEDs are outlined in the troubleshooting guide.

8) Status for Collision Detection Switches:

Here is an example of how the LEDs are used to indicate the status for collision detection, K1-K5 are the collision detection switches:
K1 → Status-LED (D12) is switched on green
K2 → Status-LED (D12) is switched on red
K3 → Front-LED (D11) at the bottom side is switched on
K4 → Back-LED left (D15)
K5 → Back-LED right (D16)
Later we will explore how the switches work when we build the collision detection circuit.

9) Resistors



Almost all LEDs require resistors. Resistors do exactly what their name implies - they resist the flow of current in a circuit. If we did not install resistors in the status circuit, the LEDs would get too much current which would cause an overload and damage the LEDs.

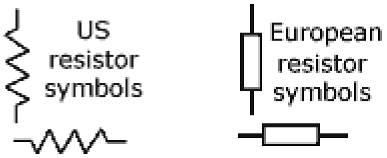
Before we can understand resistors, let's look at its opposite – conductors. What metal is the best for conducting current? If you guessed copper, you are wrong, but if you guessed gold you are right. Why is that we use copper in wires instead of gold? Of course it is cost. Gold is far more expensive than copper, because it is much more rare on Earth than copper. Conductors have many free electrons and offer little resistance to current flow. Other great conductors are silver and aluminum. But all conductors have some resistance, although it is usually very low.

How do resistors work? *If we want to make something that <u>resists</u> current flow, we probably don't want to use copper or gold or silver or aluminum. When we want to restrict current flow in a circuit, there is another material that is cheap and a great <u>material to use because it does not conduct.</u> What is it? Here's a hint. It is so cheap that we make pencils out of it.*

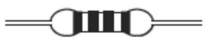
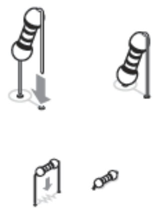
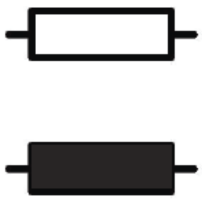
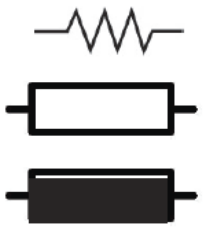
It is Carbon. Carbon is one common components used to make resistors. Another component used in resistors is ceramics, which consist of clay. Several types of resistors and what they are made of include: wirebound, carbon composition, carbon film, metal film, metal oxide film, and foil. The type of resistor we will use in the status circuit is Carbon Film.

10) Resistor Symbol

The symbol used in a circuit diagram to designate a resistor is shown here. The wavy line indicates current is being restricted in the circuit. A number used with the symbol indicates the value of the resistor.



The table below (part of table in appendix H of ARX ASURO manual) shows how to identify and assemble resistors in the ARX ASURO. The first column (component) provides a picture and indicates how to identify the component. The second column (assembly) gives directions on how the resistor is assembled or inserted into the PCB. The third column (PCB symbol) shows the symbol used on the PCB board. The fourth column shows the symbol used in a circuit diagram.

Component	Assembly	PCB symbol	Circuit symbol
Resistor/Coil 			

11) Ohms – the Measurement of a Resistor

We will install resistors in the status circuit and we will also use resistors in other circuits. The table below shows all the resistors that are used in the ARX ASURO. The amount of current that is restricted varies with the value of the resistor. We identify the value of resistors by the color bands on the sides.

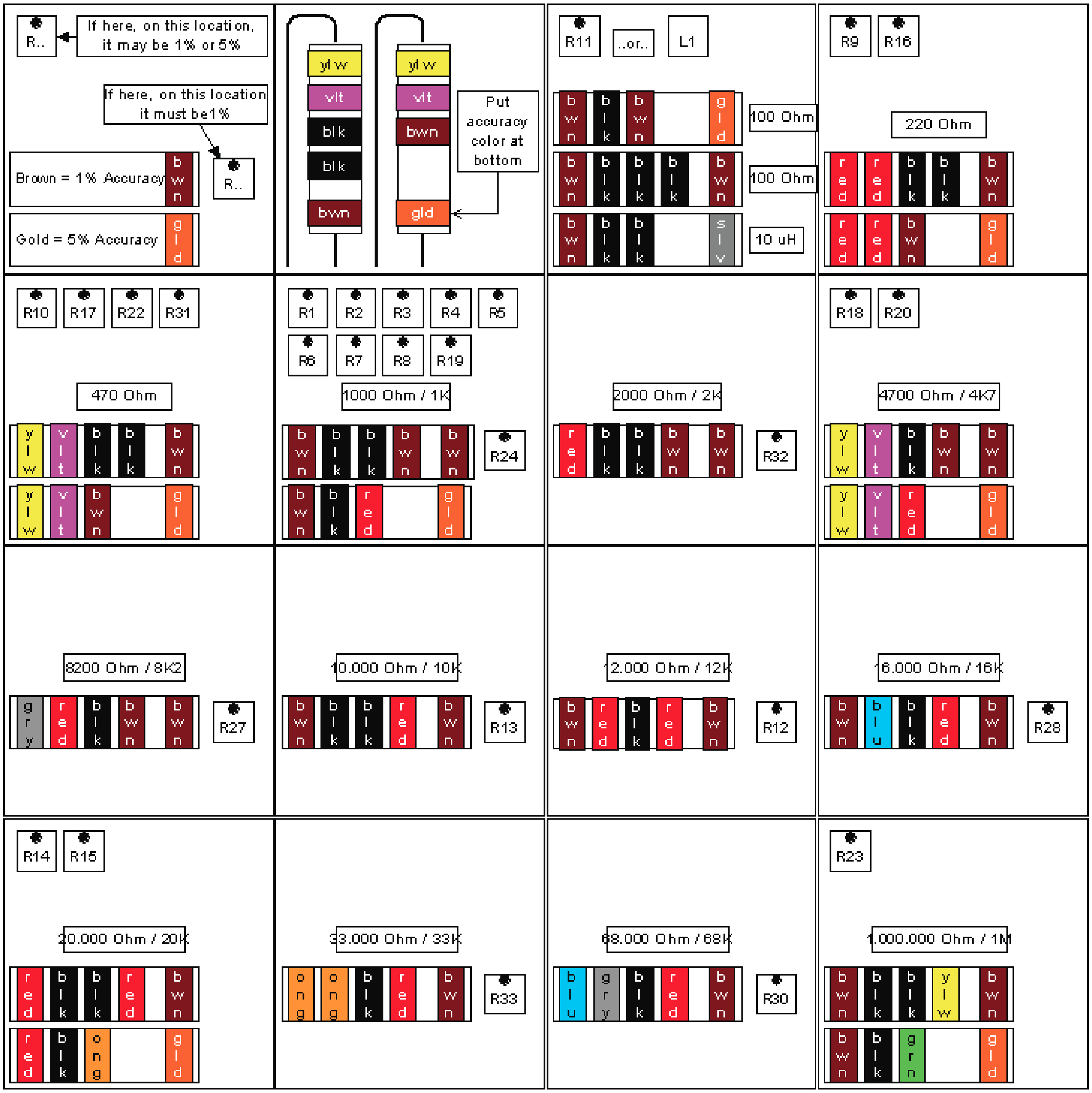
As free electrons move through a circuit they encounter atoms that do not readily give up electrons. This opposition to the flow of electrons (the current) is called resistance (R). The value or amount of resistance is determined by the number of devices which provide resistance in the circuit. Materials with high resistance are called insulators.

Resistance is measured in Ohms. The symbol for Ohms is Ω. This unit is named for the German physicist George Simon Ohm. An ohm can also be defined as the amount of resistance that will produce a potential difference (p.d.) or voltage of 1 Volt across it when a current of 1 Ampere is flowing through it.

12) Finding Resistor Values

Resistors are color coded and there is a chart which is used to associate the colored bands with the value of the resistor. For example, in the table on the next page (from the back of the ARX ASURO manual) we can look for the R number that is used in the schematics – such as R10 , R17, R22, and R31 and see that they are all 470 Ohm resistors and are identified with the colors Yellow, Violet, Brown, and Gold – in that order. We look for R9 and find that it is a 220 Ohm resistor, and is identified with red, red, brown, and gold colors. Notice that we are using the second color sets in the charts, not the first ones, because these are the types of resistors used in the U.S.

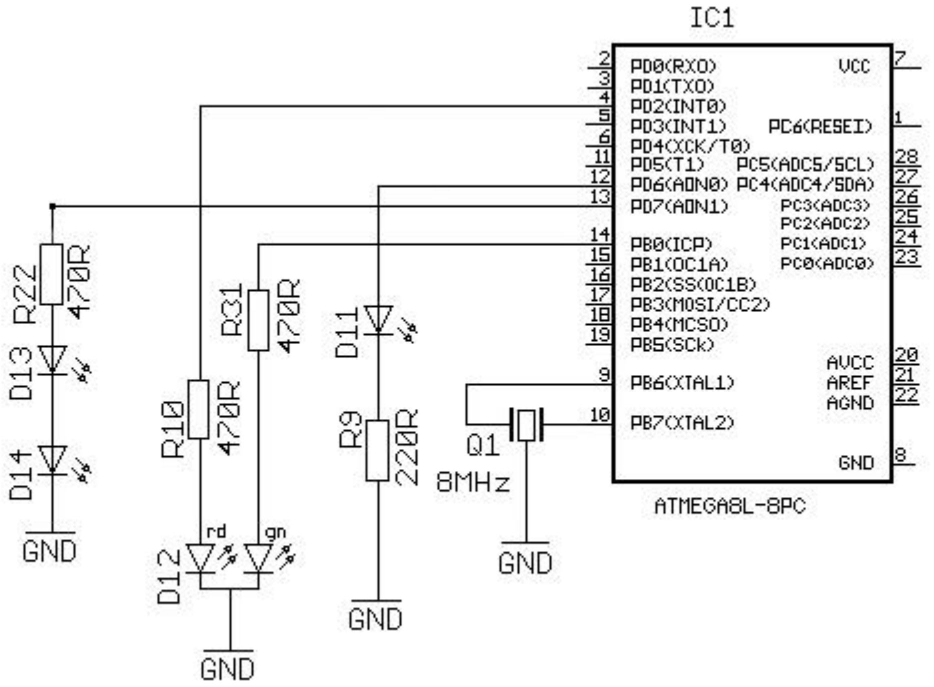
When you are working with other kits, they may not identify all the resistors for you. So it is good to become familiar with and use the standard color charts for resistors and learn how to read them. You can find them online.



Resistors used in the ARX ASURO (from ASURO manual)

B. Status Circuit Schematic

Here is the schematic drawing for the Status Circuit. We will only be building part of this circuit.



C. Parts Needed for Prototype Activity

The parts we need for this first part of our prototype circuit are:

From other kits:

- Breadboard
- Short black wire

From ARX ASURO kit:

- Power Supply
- LED D11 (red)
- Resistor R9 (220 Ohms)

The additional parts we need for the second part of our prototype circuit are:

From ARX ASURO kit:

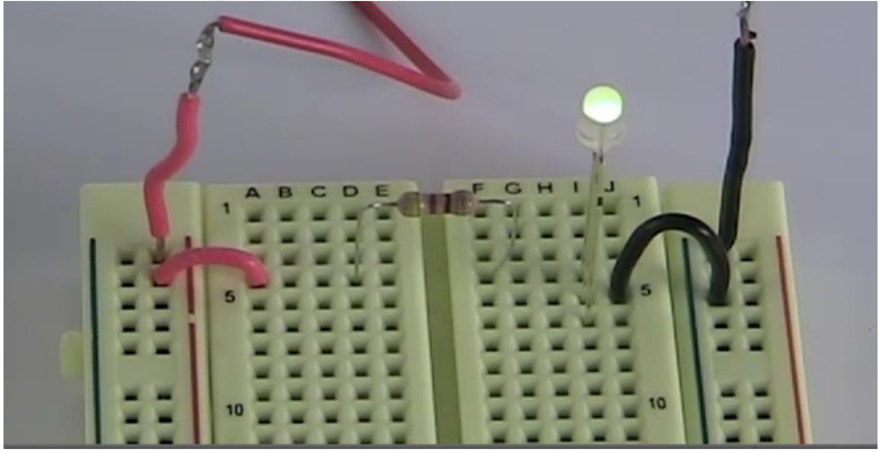
- Bicolor LED
- Two resistors (470 Ohms)

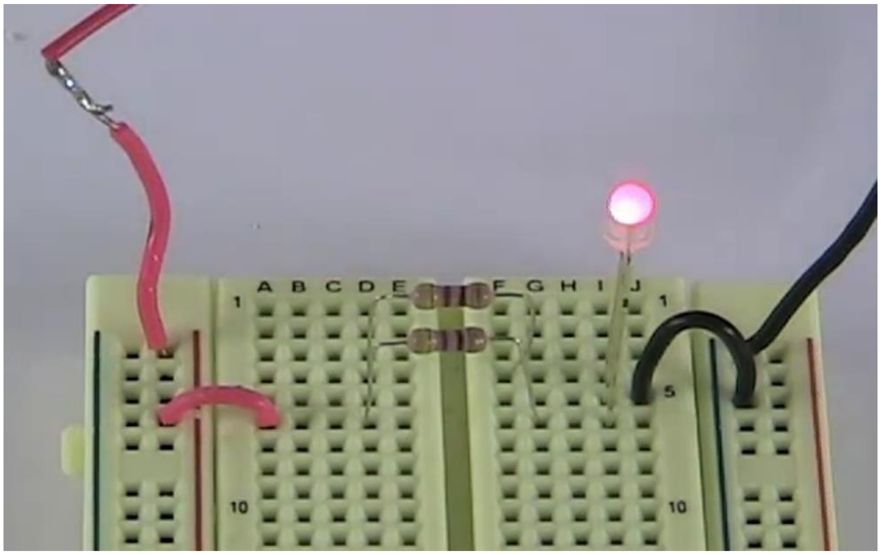
D. Status Circuit Prototype Activity

We will build two parts of the status circuit in this activity. First we will build a Red LED circuit, then we will build a bicolor LED circuit.

Assemble and wire the prototype using the breadboard, the schematic diagram, and parts as illustrated in the Status Circuit Prototype video.

Here is an example of the completed Status Circuits.





10. Collision Detection Circuit

When the ARX ASURO robot moves about a room, we want to know when it bumps into or collides with something. When a collision occurs what could we use to detect it? There are several different types of sensors that can be used, the designers of the ARX ASURO chose one type.

If you are thinking a switch you are right. One switch can detect if a collision occurs at a single point but we want to detect when a collision occurs at different points on the front of the robot so that we can react differently if the object is on the left or the right or directly in front. So we will use multiple switches, six (6) in all. With 6 switches we can tell what part of the robot has hit something.

A. Circuit Components

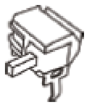
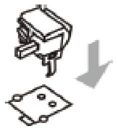
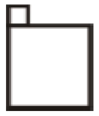
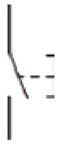
The Collision Detection Circuit contains 6 switches on the front of the robot. It also contains several resistors and some capacitors.

1) What is a Switch?

An electric switch is a device for making and breaking the connection in an electric circuit. We are familiar with switches that turn on and off lights in a room, or switches that turn on and off a device such as a TV or computer. When the connection is made (switch is on), current flows thru the circuit. When the connection is broken (switch is off), current stops flowing thru the circuit.

An electronic switch is an electronic component or device that can modify current flow in an electrical circuit, interrupting the current or diverting it from one conductor to another. There are many different types of switches, most have some mechanical method of moving from one state to another – usually a toggle or push button or key turn.

The component and circuit symbol for the switch is shown here, along with the symbol used on the PCB board and how it is assembled.

Component	Assembly	PCB symbol	Circuit symbol
Front Switch K1-K6 			

B. Series and Parallel Circuits

There are three types of resistive circuits: series, parallel, and series-parallel.

1) Resistors in Series

The more resistors connected in series, the more opposition there is to the flow of current. When resistors are connected in series, we add the resistor values together.

2) Resistors in Parallel

When resistors are connected in parallel, it is different. Each path for current to flow is called a branch. The more resistors are connected in parallel, the LESS opposition there is to current flow. The total resistance in the circuit decreases because there are multiple paths for the current to flow. Think of branches in water flow. When you have multiple branches, more water can flow thru a channel than if it only has one single branch. It is similar with current flowing thru a circuit. The total resistance is always less than the resistance of any one branch. The formula used in a parallel circuit is $1/R_T = 1/R_1 + 1/R_2 + 1/R_3$

3) Resistors in Series-Parallel

When we have a series-parallel circuit, we first calculate the parallel part of the circuit and make that one resistor value. Then we calculate the series part of the circuit and add in the resistor value from the parallel part as if it was one resistor.

4) Voltage Drop Across Circuits

In our circuit we have the switches in parallel- each of which has an associated resistance. These resistances are in series with a 1000 ohm resistance. This is the actual voltage-divider used in our circuit. Therefore if first we define the parallel resistance we can then calculate the total resistance and determine the voltage drop across both the parallel as well as the series portion of the circuit. Note we have selected each resistor in series with the sensing switches to assure that we have unique values of resistance regardless of which switches are pushed.

Before we continue, we must first understand series circuits and how to calculate the voltage drop across them - called voltage divider circuits. (*Follow the link in the activity which will explain this concept and the calculations*).

C. Logic for Switches

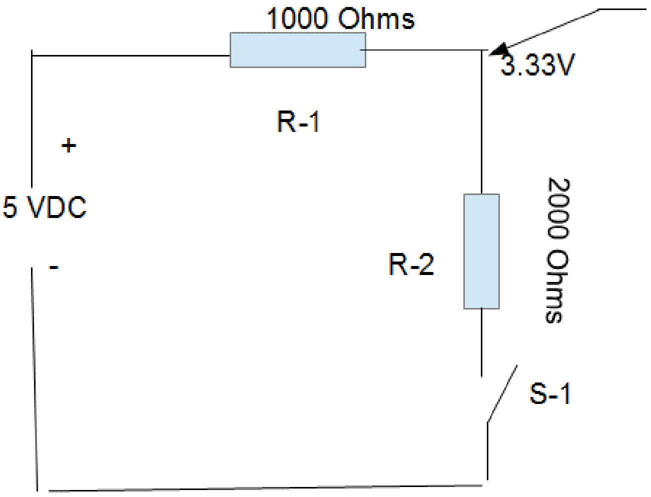
Remember our microcontroller (or brain) has the ability to sense two basic types of signals, on and off, a logic 1 or a logic 0, or a level of voltage. So if we use a logic level input for sensing collisions then we need 6 inputs for sensing, one for each switch. That would be fine if we had enough inputs on the chip, but we cannot spare 6 inputs so we must get very clever.

1) Interrupt Inputs

Interrupt inputs are a very special type of input. Just as we humans often interrupt each other, we can also interrupt a microcontroller. When a microcontroller is busy we can interrupt it by using an interrupt input. The microcontroller will stop what it is doing, save its work, and do something different, following a special interrupt routine that we write in a program. Interrupts are designed to happen very quickly so that we can handle special situations such as bumping into something.

Thus an interrupt input can be used to warn our microcontroller that we have bumped into something. But which of our 6 switches caused the interrupt? We will look more closely at interrupt sensing in a minute, but first let's consider voltage level sensing.

2) Clever Voltage Level Sensing



One method for changing a voltage signal is called a voltage divider. This method uses a number of resistors in series to vary the voltage. Let's look at this more closely. When resistors are connected in series the applied voltage is dropped across each of the resistors depending upon its resistance.

In the series circuit example diagram above, when the switch is closed, the total current is equal to 5VDC divided by the total resistance. According to Ohms law, Current = Voltage divided by resistance. We have two resistors in this series circuit, so 1000 + 2000 = 3,000 Ohms of resistance. So the calculation is Current = 5 Volts / 3,000 Ohms = 0.0016666 amps.

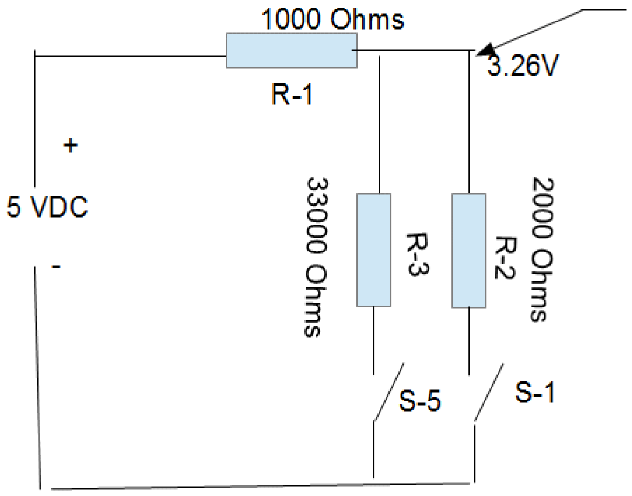
But the computer is not reading current, it is reading voltage. We measure the voltage at the place in the circuit where the A/D converter is placed (see arrow in diagram).

The voltage drop across the 1,000 Ohm resistor is equal to the Current x Resistance. Voltage = 0.0016666 amps x 1,000 Ohms = 1.6666 Volts. If we subtract 1.6666 Volts from our original 5 Volts, we have 3.334 Volts.
So our voltage at that point in the diagram is 3.334 Volts. So when Switch one is pressed, we should see a reading close to 3.334 Volts on pin 27 which is the A/D converter.

3) Parallel Resistors for a Voltage-divider

Now that we understand how a series voltage-divider works we can take the next step and consider how a parallel resistor circuit works. When we combine resistors in parallel we get different values than when we combine them in series.

In parallel the values are treated as follows. (See diagram below)



4) Parallel Resistance Calculation

To illustrate how the circuit works we must first calculate the equivalent values of the two resistors in parallel. The two resistors are those connected to S1 and S5: 2000 and 33000 respectively. The formula for calculating resistors in parallel is:
$1/ R_T = 1/R_1 + 1/R_2 + 1/R_3$

The equivalent resistance of these two resistors is $1/R_T = 1/33000 + 1/2000$. It is easiest to calculate these values using a calculator. Thus 1/33000 = 0.000030 and 1/2000 = 0.0005 adding these two together we get 0.0005303030303 Next we divide this result by 1 and we get 1/0.0005303030303 = 1,885.725 Ohms for R_T.
So our total resistance (R) of the two resistors in parallel is equal to 1,885 ohms (we ignore any fractional resistance in this case.)

Instead of having to do the math by hand for parallel circuits, fortunately there is a quick calculator online that allows us to simply input our resistor values and then it does the calculation for us. The link is included in the activity.

5) Series-Parallel Resistance Calculation

Next to calculate current in our series-parallel circuit we must add the parallel resistance to the other resistors in series in the circuit which is 1000 ohms. 1000 + 1885 = 2885 Ohms.

Ohms law calculation for current is Current = Voltage/Resistance. Assuming the voltage from our batteries is 5 volts, Current = 5 Volts /2885 Ohms
Current = 0.00173310225303 Amps

In this example, when both switch S-1 and S-5 are closed, the total current is equal to 5V DC divided by the resistance.

To find the current, we take our calculated resistance of 2,885 Ohms and we divide that into the voltage. So the calculation is Current = 5 volts / 2,885 Ohms = 0.00173 amps.
Note: If we knew the current and the resistance, we could calculate the voltage with this formula, Voltage = 0.001733 amps x 2885 Ohms or Voltage = 4.999 Volts

6) Calculating the Value the A/D converter will read

Next, we have a point in our circuit where the A/D converter will sense the voltage. We want to calculate this voltage, which we call the voltage drop across the resistors. To calculate the voltage drop across a resistor, we rewrite Ohms law as (Voltage) = (Current) x (Resistance).

Using the formula Voltage = Current x Resistance, the voltage drop across the 1,885 ohm resistance (the value of the two resistors in parallel) = current x resistance, V = 1,885 ohm x 0.00173 amps, so V=3.26VDC. So the voltage that will be sensed by the Analog to Digital converter at the point shown in the diagram is 3.26 Volts DC.

Thus if we do similar calculations for all possible combinations of switches and resistors we will find that each combination results in a unique Voltage value that allows us to determine which switch has been activated by measuring the voltage at this point with our analog to digital converter on the microcontroller.

7) Voltage Instead of Resistance

So now we know that we have a different resistance depending upon which switch(s) are depressed. But we are not measuring the resistance with our brain or microcontroller, but instead we are measuring voltage. We know however that the voltage divider formed by the parallel resistors with switches form a voltage divider with the 1K ohm resistor. Since the resistance value is always dependent upon which switches are pushed we also then know that we will have a voltage which varies with switch closures.

8) Analog To Digital Convertor

We have an analog to digital convertor as part of our microcontroller which will allow us to measure voltage in a similar fashion to the manual readings we can take with a multimeter. The analog to digital convertor is fairly accurate with a resolution of about one thousandths of the total possible reading. So if we are reading 5 volts full scale then our smallest possible reading is equal to 5/1000. Or about 0.005 Volts. This is more accuracy than we actually require but it is nice to have the potential available.

We will cover how we take a voltage reading with our built in analog to digital convertor later in the programming portion of the course. For now, just know that a program can read the values and determine which switches are depressed.

D. Parts Needed

The parts needed to build the Collision Detection Circuit Prototype are:

From other kits:

- Breadboard
- Several small black wires (from prior activities)
- Alligator clips – red and black
- Push Switches for breadboard – K1,K2,K3, K4, K5, K6

From ARX ASURO Kit:

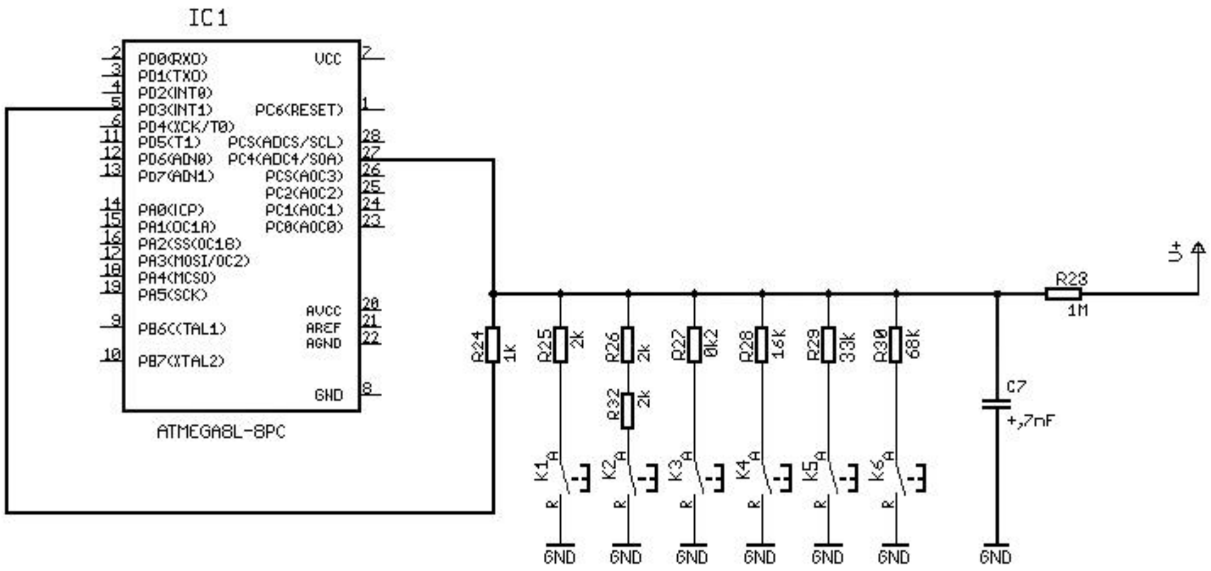
- Battery holder with batteries
- Resistors – R23, R24, R25, R26, R32, R27, R28, R29, R30
- Capacitors – C7 (4.7 nF)

Tools:

- Digital Multi-meter
- Small pliers

E. Schematic Drawing

The schematic for the collision detection system is shown below. This is one part of the overall diagram for the ARX ASURO printed circuit board. Notice that it only has one line going to the input of the microcontroller, and one line going to the A/D converter. The designers have figured out a way to send a signal to the microcontroller which tells it which switch (or combination of switches) has been depressed using only one input.



1) Switches and Resistors in Circuit

Six switches labeled K-1 through K-6 located at the front of the ARX ASURO robot are used to detect collisions. Each switch is connected in series with a resistor and then to ground. The values and labels of these switches and their corresponding resistors are:

Switch	***Resistor***	***Ohms***
K1	R25	2K
K2	R32 &R26	2K (each)
K3	R27	8K
K4	R28	16K
K5	R29	33K
K6	R30	68K

Each of these resistors is parallel with each other and is tied to the same conductor which terminates at one end with C7 a 4.7 nano-farad capacitor and a 1Meg Ohm resistor R23. The opposite side of R23 is tied to V+. The opposite end of the conductor connecting the resistors in parallel is connected to one end of R24, a 1K ohm resistor. The other end of R24 is connected to Pin 5 of the Microcontroller which is initially set as an interrupt input.

2) Circuit Activation

When the interrupt occurs, because a switch is pressed, the interrupt program, called an interrupt routine, is executed. The Interrupt input is turned into an output and a high, (1) is output to Pin 5. This causes a positive signal to be placed on R 24 and then on each of the parallel connected resistors and switches.

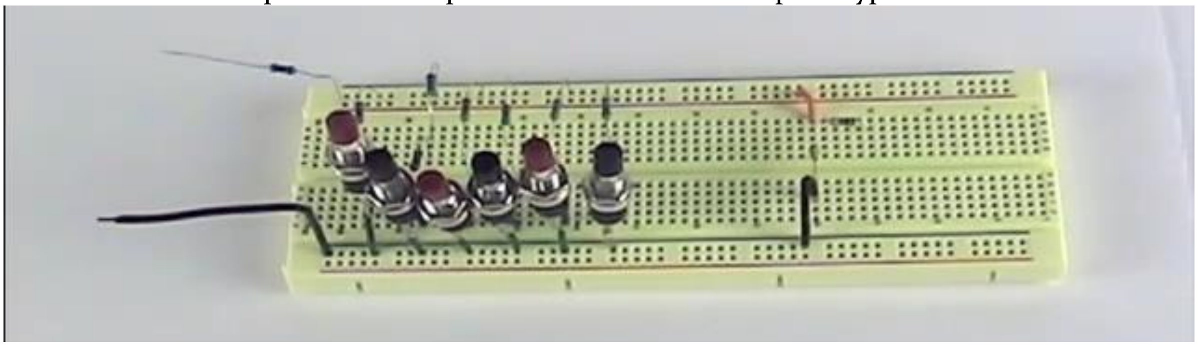
A variable voltage will then appear on Pin 27, which is an analog input ADC4. The value of the voltage depends upon which switch or switches are pressed. The program running on the microcontroller can check the value of Pin 27.

This is a clever circuit that allows the microcontroller to determine which switches are pressed based upon the voltage level sensed on Pin 27 - the analog input. This is made possible by the variation in the resistor values for the switches, which means that each switch and combination of switches pressed will yield a different voltage level.

F. Collision Detection Prototype Activity

Assemble and wire the prototype using the breadboard, the schematic diagram, and parts as illustrated in the Collision Detection Prototype video, and described in the activity.

Build a prototype with a breadboard – wire the circuit and verify all parts work. Then test the voltage sensed by computer with digital multimeter. Here is an example of the completed collision detection prototype:



11. Motor Control Circuit

Two motors provide tractive force to move and steer the ARX robot. The two motors drive a set of gears which turn the drive wheels to cause motion. The microcontroller can control both the speed and direction of each motor. These motor control functions are derived from the motor control circuit also called an H-bridge. The devices that create this circuit control on/off, speed, and direction of rotation of the motors.

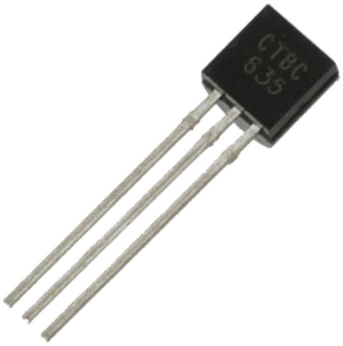
A. Overview

The ARX ASURO robot has two motors located on either side of the robot. This configuration is used to control the robot's motion. The robot will drive forward in a straight line if both motors turn at the same speed. The robot will turn in a slow arc if one motor turns slightly faster than the other. Or the robot can turn very quickly by having one motor turn in a forward direction and the other turn in a backward drive direction. These actions are accomplished with a combination of the circuits controlling the motors and the programs that control the robot.

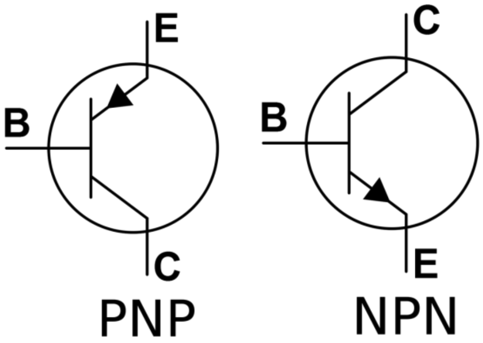
This circuit contains diodes, resistors, transistors, and integrated circuits. We have used diodes and resistors already, so we will do a quick review of transistors and AND Gates. The circuit wiring also creates a very clever device called an H-Bridge which we will also describe.

1) Transistor - A special type of switch

In a previous activity we created a circuit using switches, in this activity we will use a special type of switch; a transistor. A transistor is a three lead device.



There are different types of transistors but for now we will look at a PNP transistor.



PNP stands for Positive, Negative, Positive.
NPN stands for Negative, Positive, Negative

This tells us what connection is made to each of the three leads or wires that come out of the transistor. The three leads are called Collector, Emitter, and Base. The Base is the center lead and is used to turn on the transistor switch. When a base current is present of a very small value then the transistor turns on making a circuit path from the collector to the emitter.

This is what makes the transistor such a valuable device. A very small base current can control a much larger current in the collector emitter circuit. As an example an amplifier is an electronic device that takes a small signal and makes it much larger, like in an audio amplifier which is a part of all audio devices such as radios, televisions, and cell phones. Almost any electronic device that we listen to has an amplifier in it. Without amplifiers our electronic devices would be of little value to us. Transistors make excellent amplifier devices. This is why when radios first became popular they were often referred to as “transistor radios”.

2) More about Transistors

In this subsystem we have used four transistors of two types; PNP and NPN two of each. Now you will be able to begin to recognize transistors of these types when you see them in other circuits. But not all transistors look like these. There are a great many different types of packages and sizes for transistors, but the symbols remain similar and they also operate in similar fashion.

Major Differences
Some of the key differences include;

Appearance
Here are some other types of transistors:

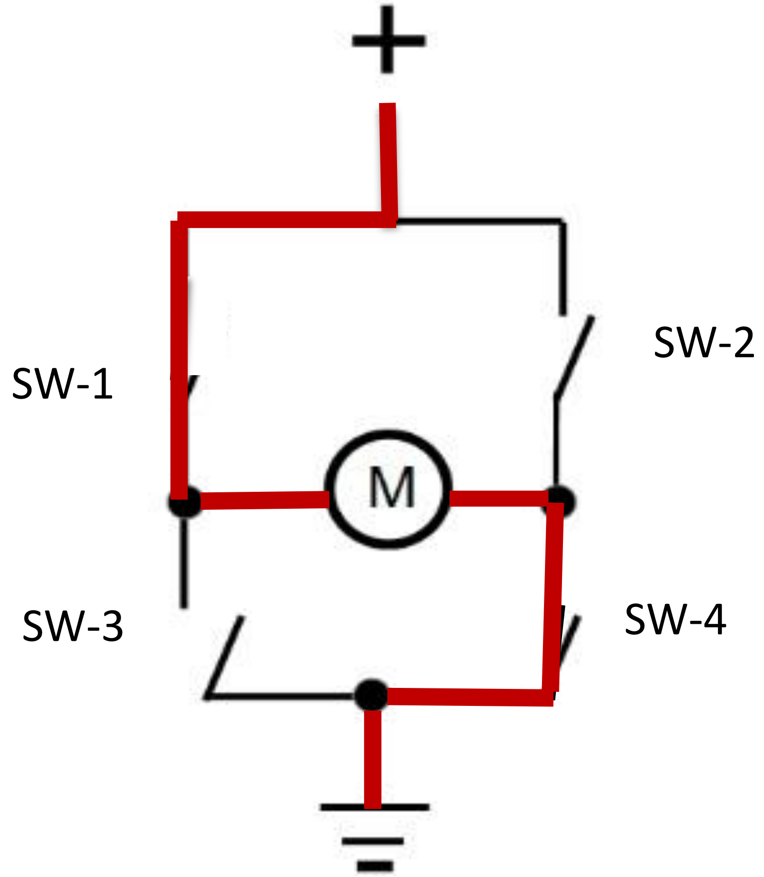
Current carrying capacity
Current carrying capacity ranges from a few milliamps to tens of amps. You can view online specification sheets to see what the range is. In addition to data sheets you can also view parameters such as current and voltage on supplier web sites. Sites to examine include Newark, Digikey, Sparkfun,

Mouser to name a few. Or simply type in "transistor" in your favorite search engine and view the search results.

Amplification Factor (hFE)
Transistors have a wide range of hFE or amplification factors from as low as less than 20 to 2 or three hundred. Once again view data sheets and search online for hFE.

Operating Voltage
Voltage can range from a few volts to several hundred.

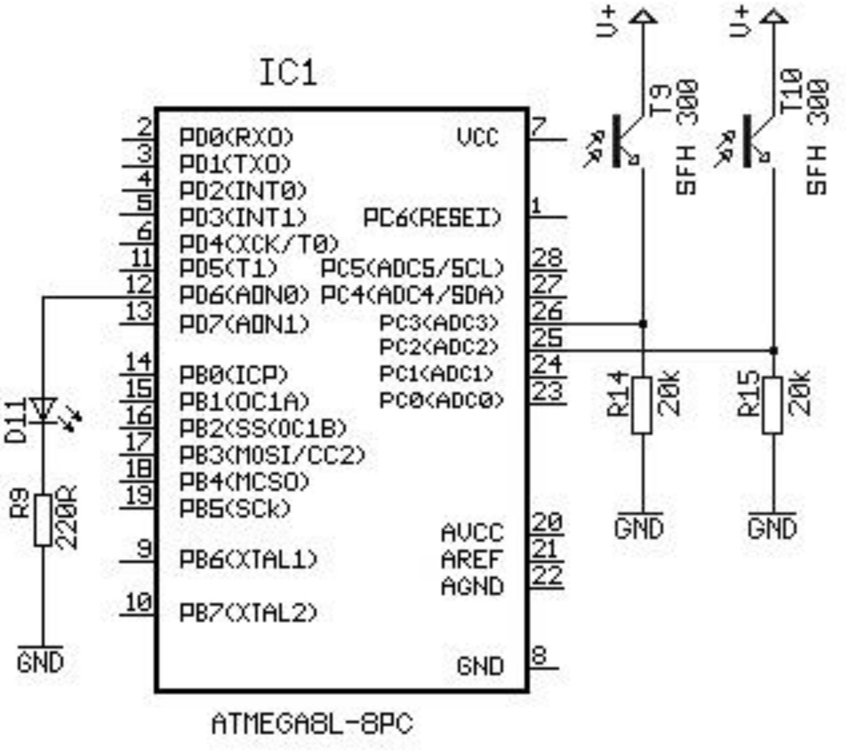
Heat Dissipation
With increased current heat dissipation becomes a factor. Once again check the data sheet or search online.



H Bridge 4 Switch Motor Control Concept

3) H Bridge

In addition to recognizing transistors you have also become familiar with a popular method of controlling DC motors; the H Bridge. This is a common circuit that is widely used. In addition to building your own circuit as we did in this case you can also purchase an H Bridge as a prepackaged unit, or device.



The Motor control Subsystem is made up of a circuit which is *commonly called an **H Bridge***, because of the way in which the circuit is configured, we will explain this later in our discussion. There are actually two similar almost identical circuits one for the left and one for the right motor. We will describe only one of these circuits, the left and leave the other for you to review on your own.

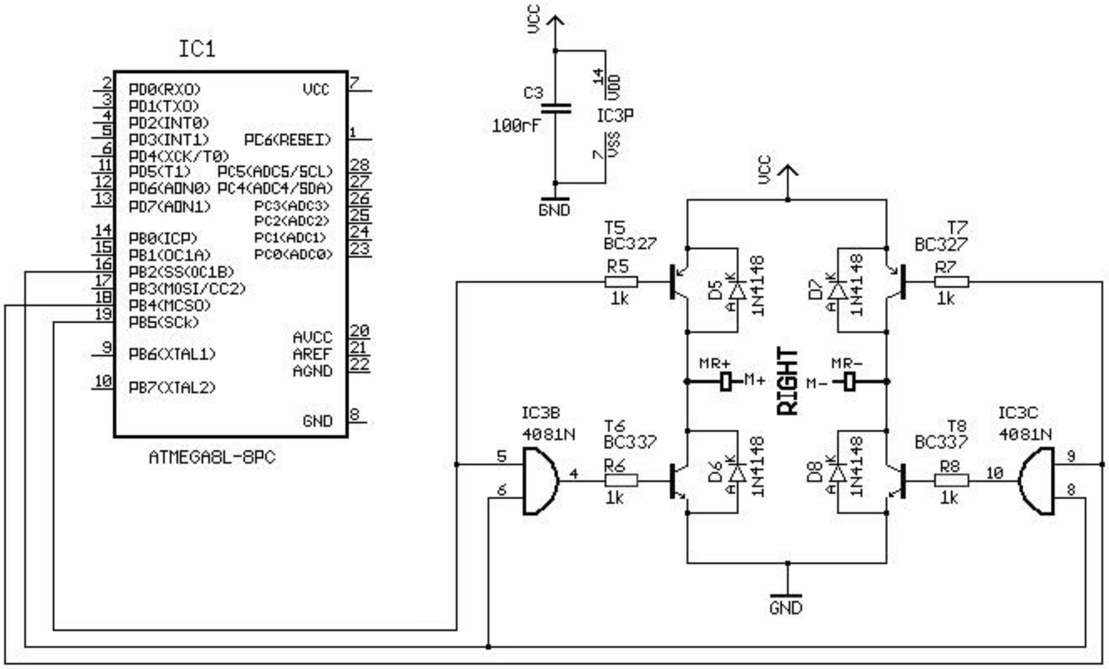
To turn the motor, use two switches. Closing switch 1 and switch 4 will cause the motor to rotate in one direction - for example clockwise.

In our next situation we close switch 2 and switch 3. Now the positive power is applied to the right side of the motor and the motor will run in the opposite direction, or counter-clockwise.

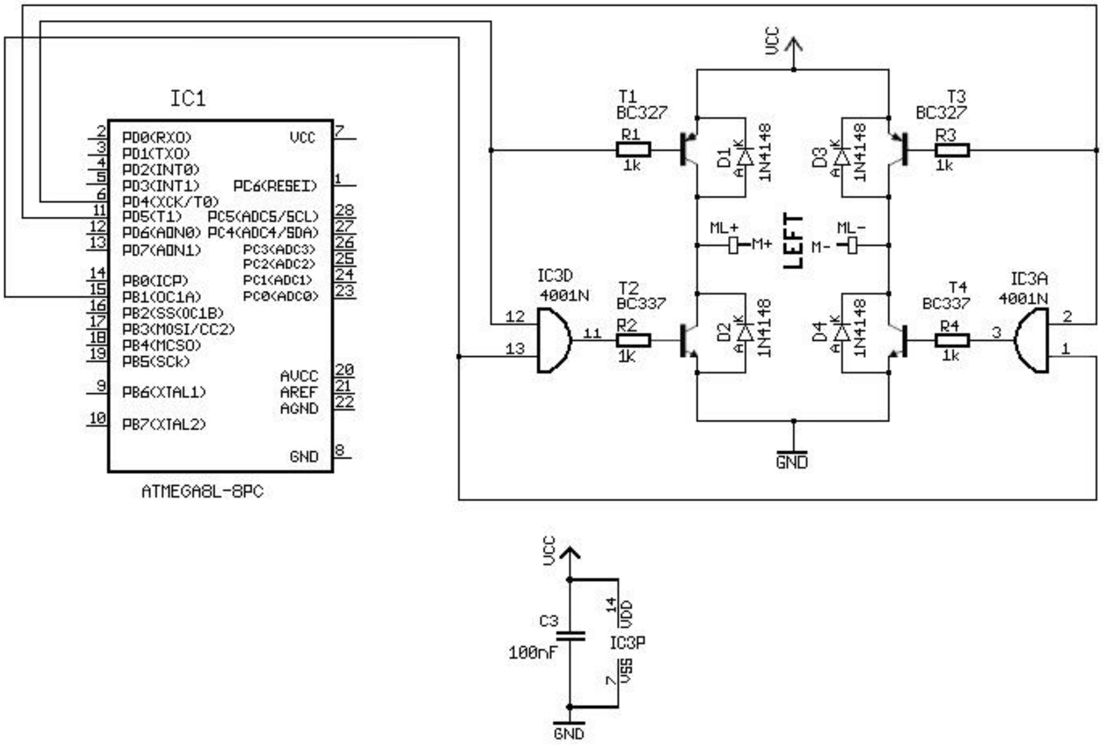
B. Schematics

Our microcontroller is not able to output a very strong signal by itself, such as required for turning a motor on and off. But if we use a transistor, we can amplify that small signal and control our robot's motors. When we build the circuit and test it, we can see how this works. Here are the schematic diagrams for the right and left motor control circuits.

Right Motor Control:



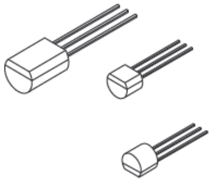
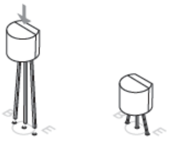
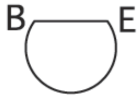
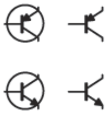
Left Motor Control:



1) The Transistors

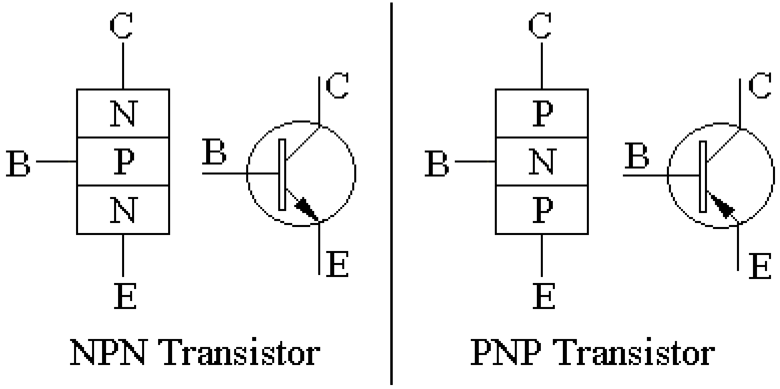
The circuit uses two PNP transistors, and two NPN transistors. Basically a transistor is a special type of solid state switch. Each of the transistors used here is turned on by applying a very small current to the base of the transistor (one of its legs or leads).

Each transistor used here has three parts an Emitter, a Base, and a Collector. The essential way in which a transistor operates is that a potential voltage is applied to the collector and emitter and when a small current is allowed to flow in the base the transistor will turn on allowing a much larger current to flow in the collector-emitter circuit. Note the flat side of the transistor is used to orient it to tell which leg is connected to the positive and which is connected to the negative.

Component	Assembly	PCB symbol	Circuit symbol
Transistor 		*B and E are just an example, position can change per type*  	*PNP type* *NPN type* 

2) PNP and NPN Transistors

The two types of transistors used here (PNP and NPN) differ in how the collector and emitter are wired to the power supply. The PNPs are part number BC327 and the NPNs are part number BC337. It is important that they are wired and inserted correctly or they will not work. In the PNP, the emitter is connected to the positive source of power. See the symbol below for the PNP to identify the three leads.

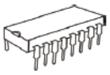
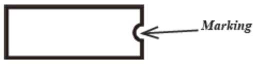
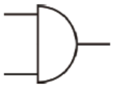


NPN vs PNP Transistors (C= collector, B= base, E=emitter)

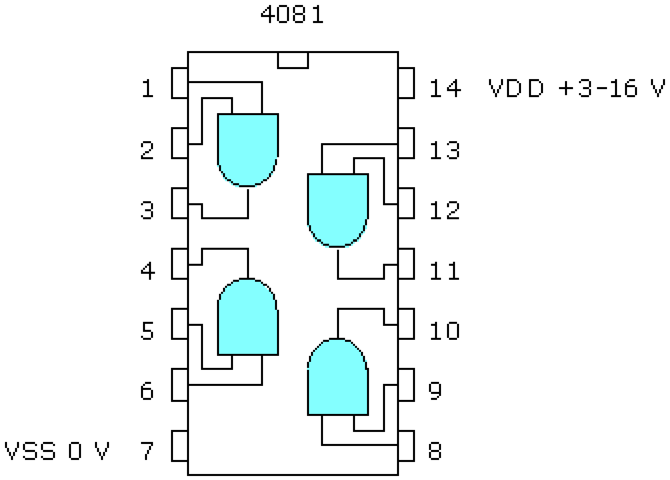
Look at the schematic and locate T1, which is a PNP transistor. Note that the emitter for both T1 and T3 are connected to VCC which is the positive side of the power supply. Only one of these two PNP transistors will be turned on at any time. Assume that a small negative signal is applied to the 1K ohm resistor, this will cause a very small current to flow in the Base circuit turning on the transistor. This small negative signal is obtained from a Low (0) on Pin 6 of the Microcontroller.

3) Using the AND Gate to Turn the Motor Forward

This same negative signal is applied to one of the input pins of IC3D, an AND Gate (Part Number CD4081).

Component	Assembly	PCB symbol	Circuit symbol
IC CD4081 	Notice correct polarity!		

As shown in the diagram below of the IC 4081, it contains four AND gates. The numbers shown on the lines in the AND gate symbol in the Motor Control Schematic Diagrams correspond to the pins on the IC shown here.



Each AND Gate has two inputs and one output. Note that it takes a positive or high on each input to get a High(1) on the output, thus a negative input on the AND Gate (IC3A) assures that the output is Low thus assuring that T2 (a NPN transistor) cannot be turned on. This is important because if both T1 and T2 were energized at the same time, a direct short would occur when a path is provided between the positive and negative side of the power supply.

Instead we must turn on T4. This is accomplished by applying a High(1) to both inputs of the AND Gate (IC3A). We accomplish this by applying a High (1) on Pin 15 of the microcontroller and also on Pin 11. This turns on T4 and completes the circuit for the motor.

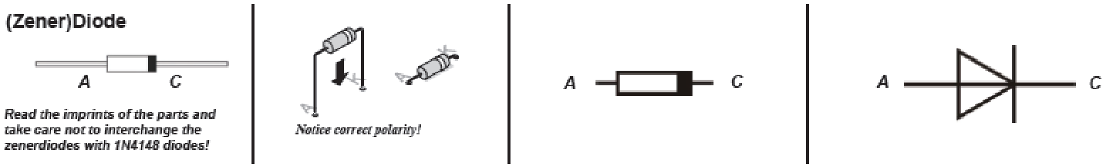
4) Using the AND Gate to Turn the Motor in Reverse

In order to reverse the motor, we must reverse the power to the motor leads by turning off T1and T4 and energizing T3 and T2. T2 is energized by applying a negative signal to the base of T2 through the 1K ohm resistor by outputting a Low(0) on Pin 11. This also puts a Low (0) on Pin 2 of the AND Gate (IC3A) which turns off the AND Gate and assures that T2 and T4 are not on at the same time, causing a direct short.

At the same time a High (1) is applied to Pin 12 of the AND Gate (IC3D) by placing a High (1) on Pin 6 of the microcontroller. This also places a High (1) on the base of T1 assuring that it will not turn on. Concurrently a High(1) is applied to Pin 13 of the AND GATE (IC3D) by placing a High (1) on Pin15 of the Microcontroller.

5) Diodes

Closely observe the polarity of the diodes associated with each transistor, and double check their orientation. There is a black band on one end which is the cathode side. Also, don't confuse these 1N4148 diodes with the zener diodes.


(Zener)Diode
A
C
Read the imprints of the parts and take care not to interchange the zenerdiodes with 1N4148 diodes!
Notice correct polarity!
A
C
A
C

C. Parts Needed

Right Motor Control:

- Drawing and Diagram
- Transistors - T5, T6, T7, T8
- Integrated Circuit IC CD4081 – (IC3B, IC3C)
- Diodes – D5, D6, D7, D8
- Resistors - R5, R6, R7, R8

Left Motor Control:

- Drawing and Diagram
- Transistors - T1, T2, T3, T4
- Integrated Circuit IC CD4081 - (IC3A, IC3D)
- Diodes– D1, D2, D3, D4
- Resistors- R1, R2, R3, R4

Parts for building the Prototype:

From other Kits:

- Breadboard
- Four Switches for breadboard
- Red and black wire

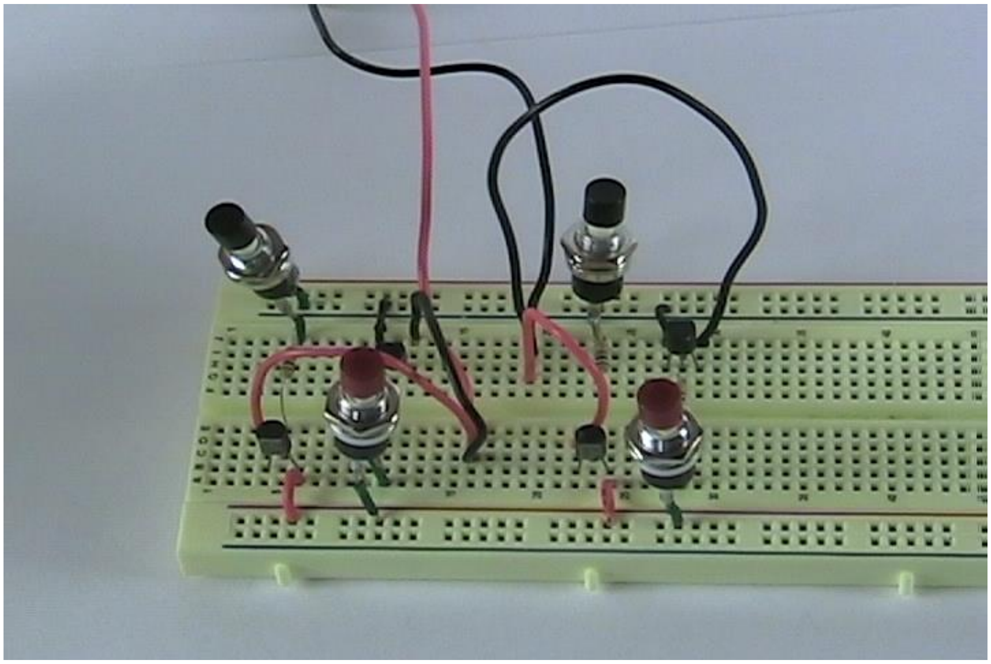
From ARX ASURO Kit:

- Power Supply with four AAA batteries
- Two DC motors
- Transistors – two NPN and two PNP
- Resistors – all 1K Ohm

D. Motor Control Circuit Prototype Activity

Assemble and wire the prototype using the breadboard, the schematic diagram, and parts as illustrated in the Prototype Assembly video. Follow the instructions in the activity description.

Here is an example of the completed motor control circuit prototype:



12. Encoder or Odometer Circuit

When the ARX ASURO moves forward or backward, how does it know the distance it has traveled or its speed? The engineers gave it a device called an Encoder. This device works with an encoder circuit to track the movement of the wheels, and from that, a special program determines the distance and speed.

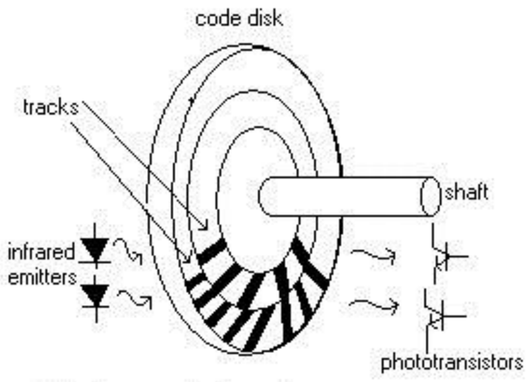
The ARX manual refers to the encoder circuit as an odometer. The logic behind this name is that the encoder circuit senses the presence or absence of a black target mounted on one of the ARX gears. As the motor turns the gear, the light and dark areas of the target are sensed with a phototransistor. Counting the number of transitions from light to dark provides monitoring of the rotation of the gear and thus the relative position of the ARX can be determined programmatically and calculate the distance traveled - hence the name odometer.

An encoder is an electronic device that converts input information to a code useful in digital circuitry. To decode means to translate from one code to another. We prefer using the term encoder as this term has far broader use in many industrial applications.

A. Encoder Circuit Components

An encoder is an electronic device that converts input information to a code useful in digital circuitry. Encoders are made up of a code disk that has a pattern of black tracks on it.

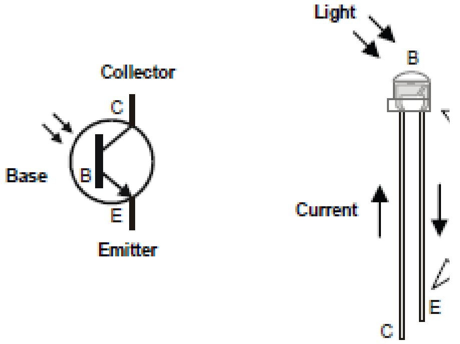
Infrared emitters or other LEDs are used to light up the tracks and Phototransistors detect the difference between the black and white areas of the track.



1) The Photo Transistor

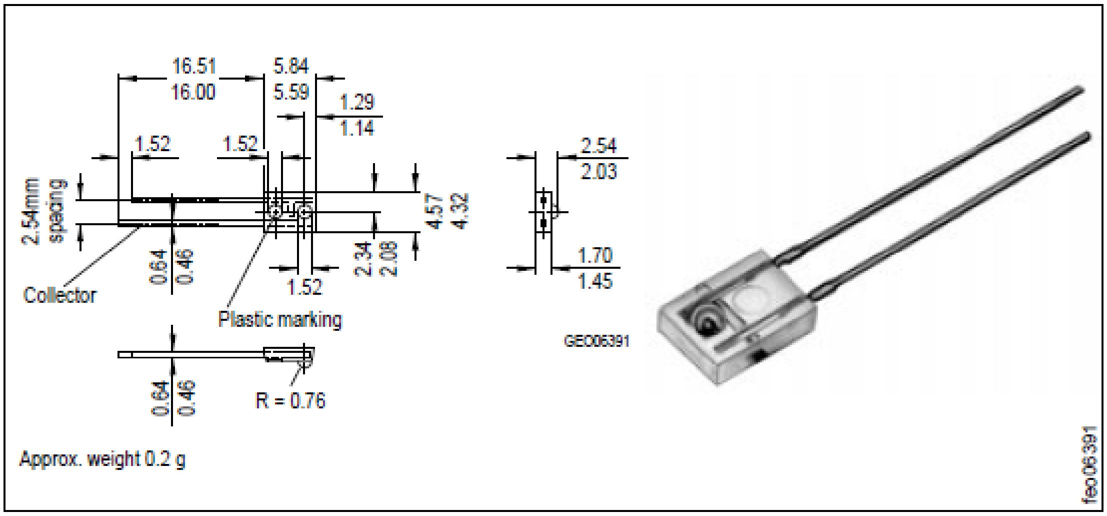
Photo transistors are similar to the transistors we have used with a very important distinction. Instead of using a base current to turn the transistor on, light is used. Thus the name photo transistor, photo refers to light. In this circuit we will see how photo transistors are used for wheel encoder sensors.

The brightness of the light shining on the phototransistor's base (B) terminal determines how much current it will allow to pass into its collector (C) terminal, and out through its emitter (E) terminal. Brighter light results in more current; less-bright light results in less current.



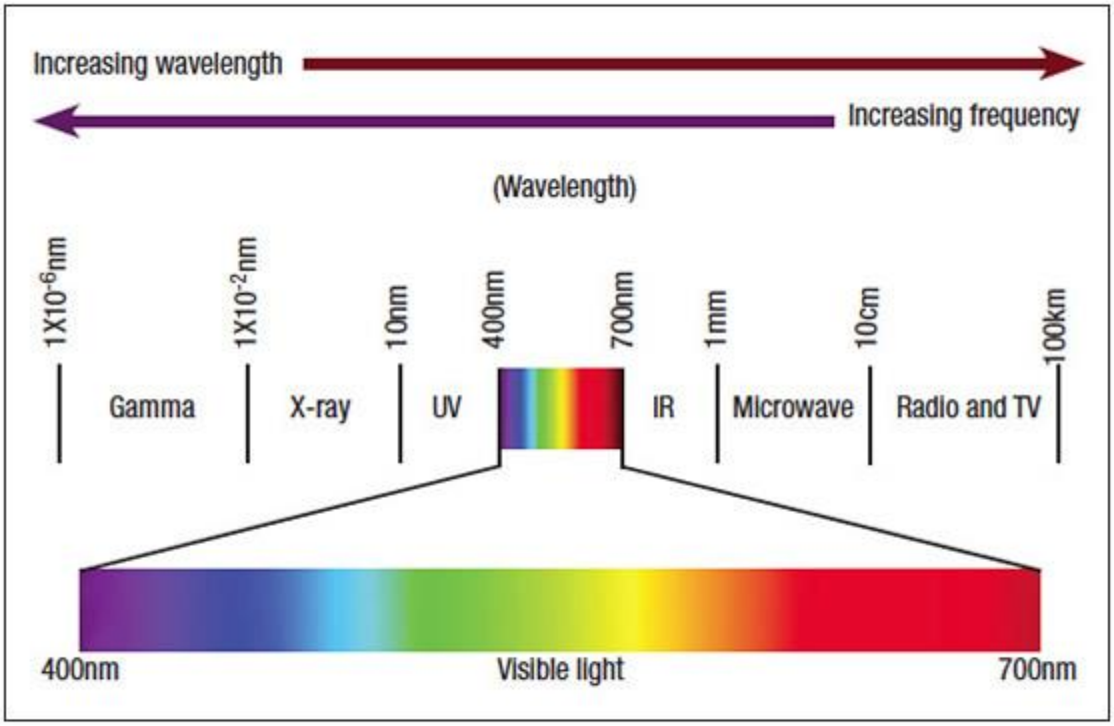
One leg of the phototransistor is slightly shorter than the other. That leg is the Emitter on this phototransistor. The phototransistor must be inserted in the correct orientation to work properly.

Light travels in waves and the distance between adjacent peaks is measured in nanometers (nm) which are billionths of meters. According to the Manufacturer's specifications, the phototransistor model LPT80A is especially suitable for applications from 470 nm to 1080 nm, which includes visible light and goes into the Infrared range (see wavelength chart). Light Visible to Humans is between 400nm and 700nm.



Specifications of the LPT80A Phototransistor

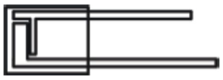
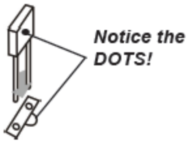
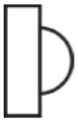
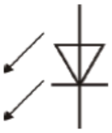
Each phototransistor has a range of sensitivity. Phototransistors are built to respond to particular waves of light – usually either in the Infrared light range or in the visible light range. The collector-emitter voltage of the LPT80A is 30 Volts, and the collector current is 50 mA.



Wavelength Chart

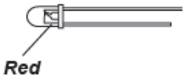
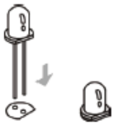
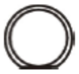
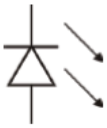
2) Infrared LEDs

The encoder circuit operates based upon light from two Infrared LEDs – (D13 and D14) which are part number IRL80A. These LEDs are pink in color and have a dot on the side which emits light. The LED shines infrared light onto the encoder wheel and the light is bounced off and picked up by the phototransistor.

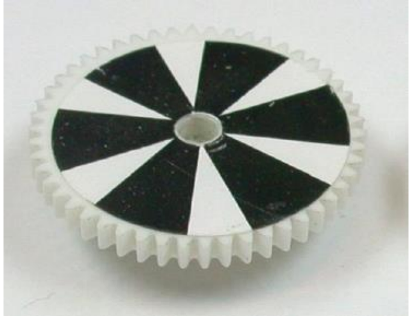
IR-LED IRL80A Side type pink color 			

3) The Red LEDS

Two status Red LEDS (D16 and D15) are used in conjunction with the photo transistors to provide a signal that the encoder is working. These LEDs flash on and off and are located at the back of the ARX ASURO.

Component	Assembly	PCB symbol	Circuit symbol
LED red D15-D16 	 C (Cathode) = Short leg should be on flat side!	 Flat side is K = -	

4) Encoder Wheel



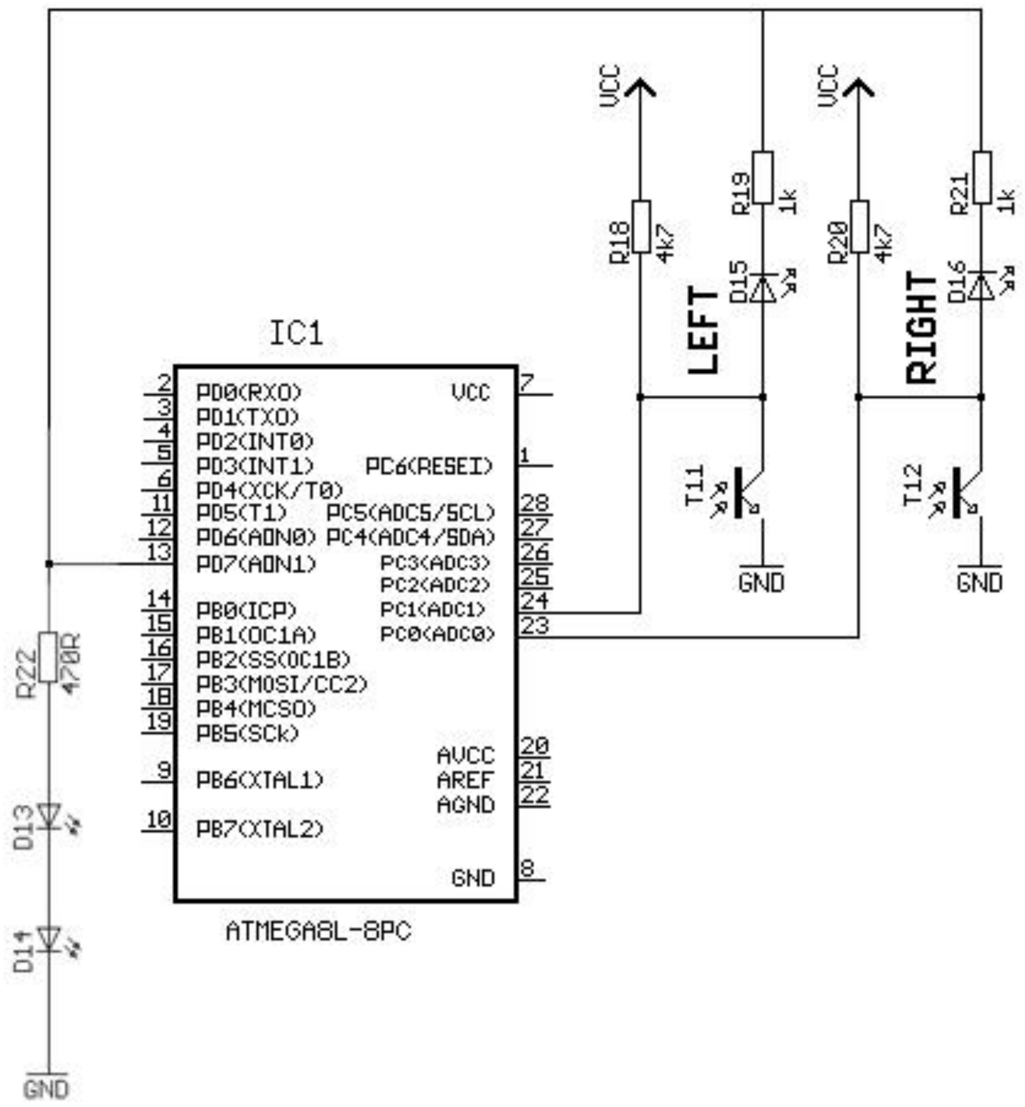
Light from LEDs falls on the two encoder wheels, (one right and one left). The encoder wheels have a sticker that is applied that looks like a target with light and dark areas. The photo transistor senses the light and dark areas as the gears rotate.

5) A to D Converter

The signal from the two photo transistors are sent to the microcontroller Analog to Digital convertor inputs ADC0 and ADC1 via pins 23 and 24. These inputs are monitored by the encoder subroutine to track the robot's position. We will explore how these devices work by building a prototype of the sensor circuit. But we will not include the encoder wheel or the microcontroller's A to D converter in our prototype. Instead we will simulate their operation.

B. Encoder Circuit (Left and Right)

Here is the schematic diagram for the left and right encoder circuit:



As we see in the schematic diagram, the encoder circuit consists of IR LEDs – D13, D14, two regular LEDs, D15, D16, Photo Transistors – T11, T12, and Resistors – R18, R19, R20, R21, and R22

1) **The IR LEDs**

The encoder circuit operates based upon light from the two Infra-Red LEDs - D13 and D14. These two LEDs are energized by current flowing through the 470 ohm resistor (R22) which is in turn energized by a High (1) on Pin 13 of the Microcontroller. D13 and D14 are in series with R22, which is a 470 Ohm current limiting resistor.

2) **The Photo-Transistors**

Photo-Transistor T11 is energized via the 4.7K ohm resistor R18. As more or less light is reflected onto T11 it conducts more or less current and thus provides a varying voltage to Microcontroller Pin 24 which is an input for ADC1 (analog to digital convertor input).

This varying voltage (which represents the passing of the white and black areas of the encoder label attached to the Driving gear) allows the encoder software program to track the movement of the motor and attached wheels, thus tracking the movement and position of the robot.

In a similar manner, Photo-Transistor T12 tracks the movement of the other motor and wheel through reflected IR light hitting its sensor. This varies the voltage sensed on Microcontroller Pin 23 (ADC0) of the microcontroller. T12 is energized by the 4.7K ohm resistor R20.

C. Parts Needed

The parts used in the circuit are:

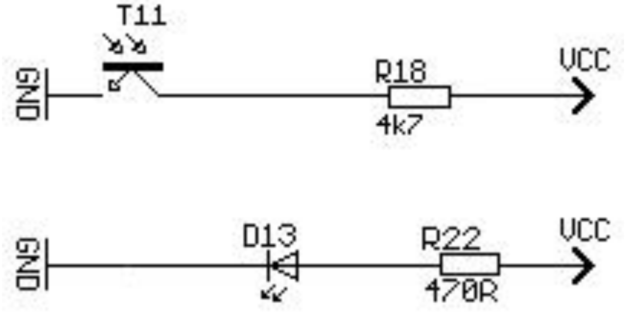
- Photo-Transistor = T11, T12 (clear)
- Resistor = R18, R20 = 4.7K (2 Reqd.)
- IR LEDs = D13, D14 (pink)
- Resistor R22 = 470 (1 Reqd.)
- Status LEDs with Resistors
    - LEDs D15, D16 (RED LEDs)
    - Resistor R19, R21 = 1K (2 Reqd.)

Parts Needed for building the Prototype:

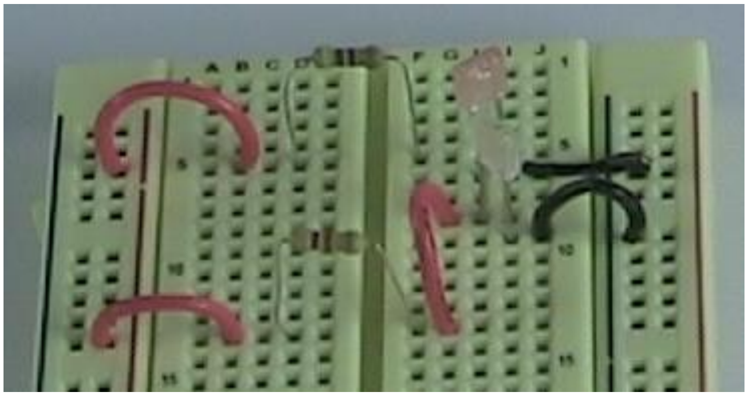
- Breadboard
- Power Supply
- Photo-Transistor LPT80A (clear and flat)
- IR LED (pink and flat)
- Resistor 4.7K ohm
- Resistor 470 ohm
- Digital Meter with alligator clips
- White paper with black stripe

D. Encoder Prototype Activity

For this activity we will build only one section of the circuit, as shown in this circuit diagram.



Assemble and wire the prototype using the breadboard, the schematic diagram, and parts as illustrated in the Encoder Prototype video. Follow the instructions in the Activity. Here is an example of the completed Encoder Prototype Circuit.



13. Line Follower Circuit

The ARX ASURO robot was designed to be able to follow a line as one method of control. The ability to following a black line on a white or light surface is common for robots used in office buildings and warehouses.

A. Line Follower Components

Line following is enabled by using three devices working together. An LED illuminates the line and two phototransistors sense reflected light to determine the presence or absence of the line. A high contrast black line against a white background is the line to be followed.

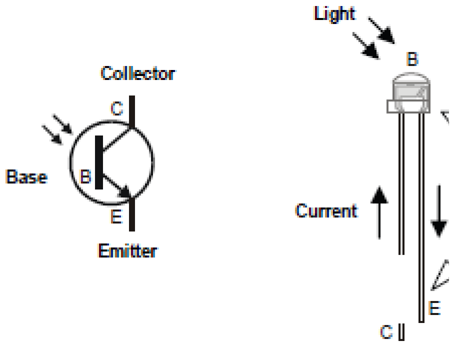
The signal from two phototransistors are sent to the analog to digital converter. These inputs are monitored by the line tracing program to track the position of the line being followed.

1) The Photo-Transistor

The photo transistors used in this circuit function similar to the ones used in the encoder circuit. But they are a different model and they look different. These photo transistors look like clear LEDs and are model SFH300.

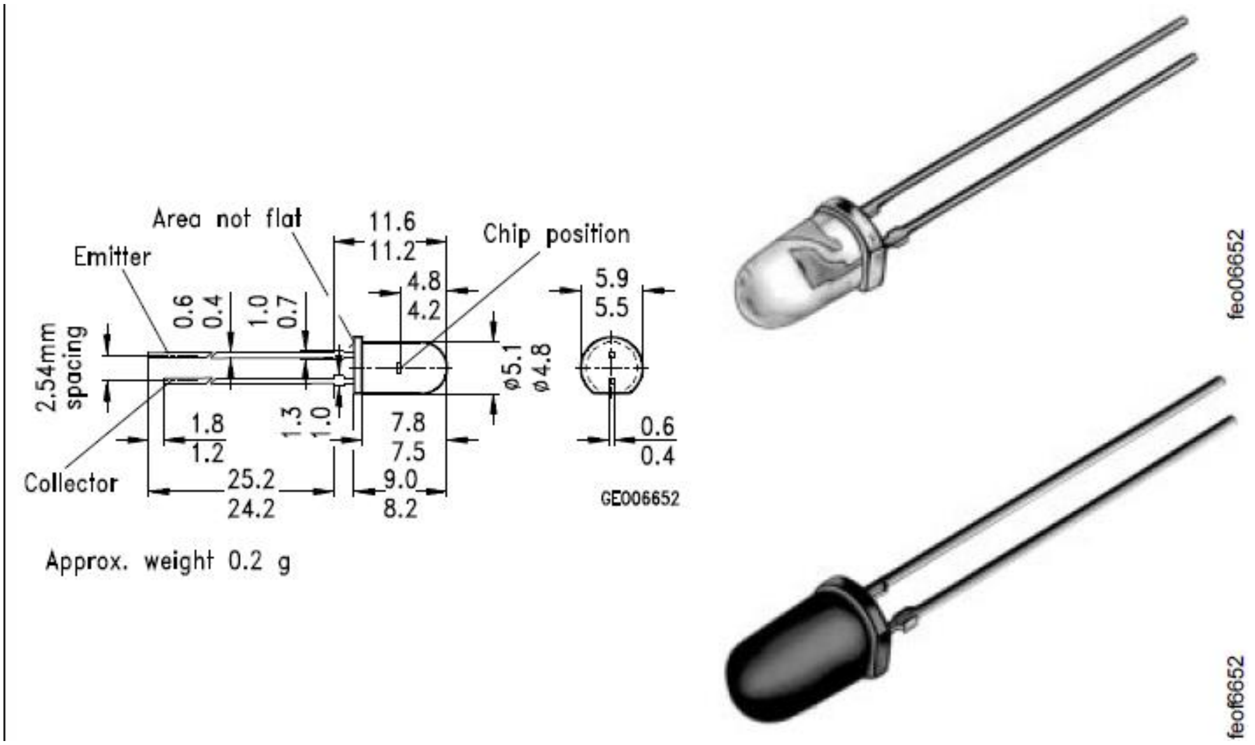
Just like the wheel encoder phototransistors, these have a Base, collector, and emitter. Instead of using a current at the base to turn the transistor on, light is used instead.

The brightness of the light shining on the phototransistor's base (B) terminal determines how much current it will allow to pass into its collector (C) terminal, and out through its emitter (E) terminal. Brighter light results in more current; less-bright light results in less current.

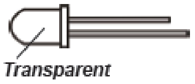
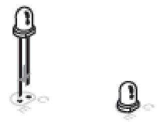
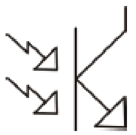


One leg of the phototransistor is slightly shorter than the other. That leg is the Collector on this phototransistor. The phototransistor must be inserted in the correct orientation to work properly.

Light travels in waves and the distance between adjacent peaks is measured in nanometers (nm) which are billionths of meters. The phototransistor model SFH-300 is especially suitable for applications from 420 nm to 1130 nm, which goes into the Infrared range (see wavelength chart in encoder circuit). Light Visible to Humans peaks at around 700nm.



Specifications of the Siemens SFH 300 Phototransistor.

Component	Assembly	PCB symbol	Circuit symbol
PHOTOTRANSISTOR SFH-300 			

2) **The LED**

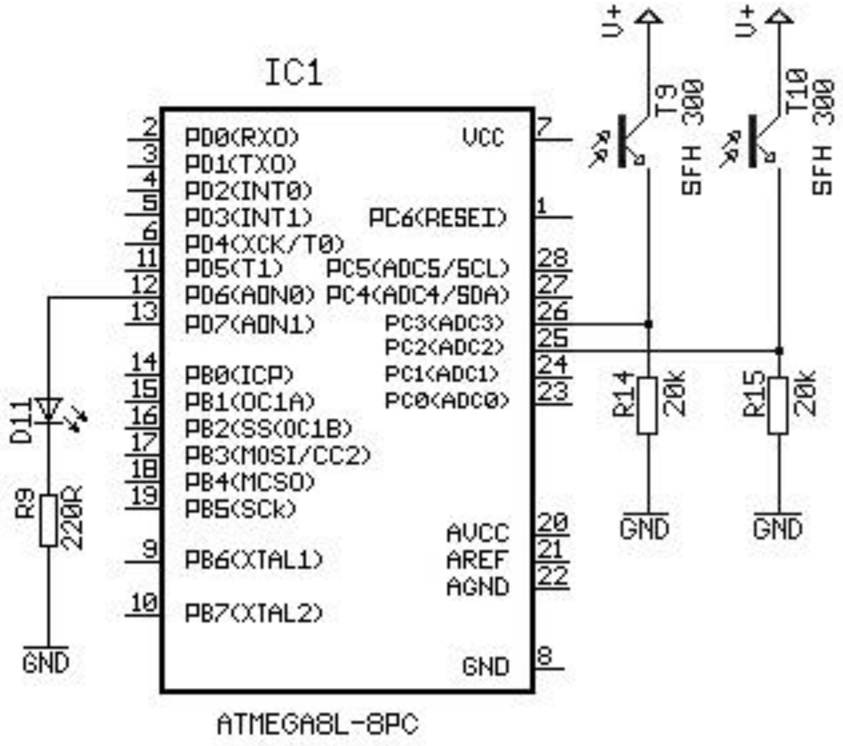
One LED is used in conjunction with the photo transistors to provide a signal that allows us to monitor the black line being followed. Light from LED D11 falls onto the black line and is reflected back and monitored by phototransistors T9 and T10 located on the bottom of the circuit board, along with LED D11.

3) **Microcontroller inputs**

The signal from the two photo transistors are sent to the microcontroller Analog to digital convertor inputs ADC2 and ADC3 via pins 25 and 26 on the microcontroller. These inputs are monitored by the encoder subroutine to track the robot's position.

B. Schematic Diagram

Here is the schematic diagram for the line follower circuit:



NOTE: Diagram does not show LED D11, it's resistor and going to Pin 12

As we see in the schematic diagram, the line follower circuit consists of these main components: two phototransistors T9 and T10, one LED (D11), and two resistors R14 and R15. When we later assemble these on the robot, they will be located at the front, underneath the robot. The LED illuminates the line to be followed. It is energized by a high (1) on Pin12 of the microcontroller.

The phototransistors are mounted on either side of the LED. Phototransistors T9 and T10 have their collectors connected to V+ and their emitters connected to ground through resistors R14 and R15 respectively, each of which is 20K ohms.

As the ARX passes to the right or left of the line being followed, the phototransistors sense the presence or absence of the line. They send a varying signal to Pin 26, (ADC3), and Pin25, (ADC2). The microcontroller's Line Following Program examines the two relative values of the input signals and determines if the robot is either to the right or left of the line or directly over it. We will review this further later in the Line Following Program.

C. Parts Needed

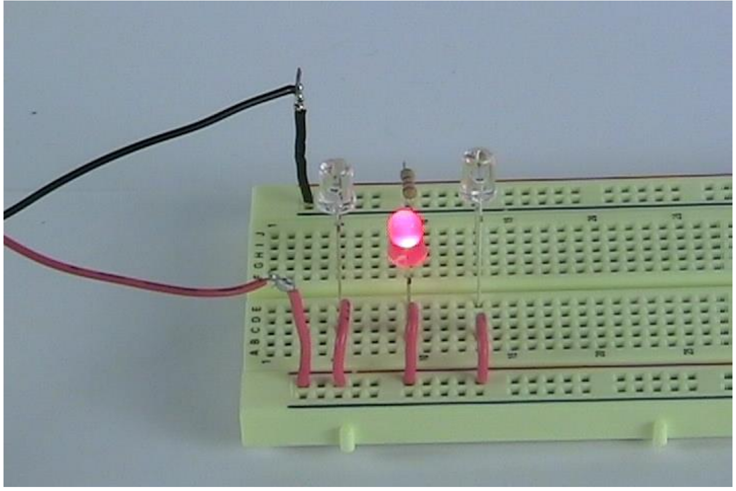
We will explore how the devices for the line follower work by building a prototype of the circuit. The parts needed to build the prototype are:

- Breadboard
- Power Supply with AAA batteries
- 2 Photo-Transistors (clear and round)
- Red LED
- 2 Resistors 20 K ohm
- 1 Resistor 220 ohm
- Meter, alligator clips
- White paper with black stripe

D. Line Follower Prototype Activity

Assemble and wire the prototype using the breadboard, the schematic diagram, and parts as illustrated in the Line Follower Prototype video. Follow the instructions in the Line Follower Prototype Activity.

Here is an example of the completed line follower prototype circuit:



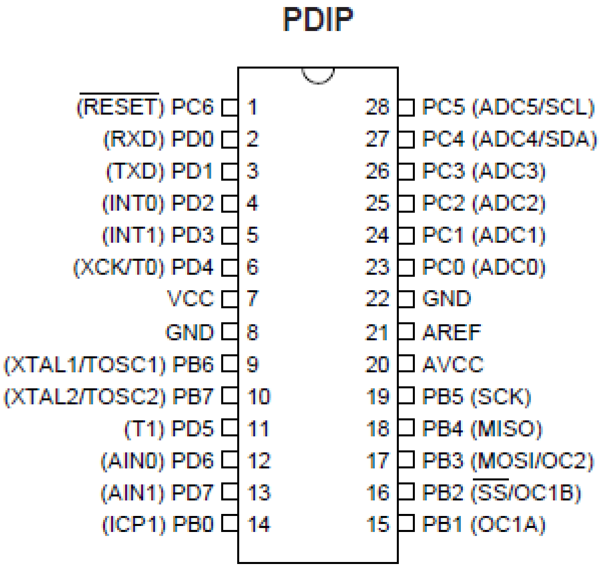
14. Microcontroller Circuit

How does the ARX ASURO robot make decisions?
The brains of the robot is an integrated circuit, the microcontroller. Programs are loaded onto the microcontroller, and it is those programs which make decisions for the robot.

But there is more to it than that. As we have seen in our circuit diagrams, each system connects to the microcontroller. It is those connections which allow the microcontroller to receive and send information and control the operation of each system.

A. Overview

The ATMega8 microcontroller is the brains of the ARX robot. It controls all the actions of the robot via digital and analog inputs that allow it to sense information about the robot's environment and operation. These digital signals are on/off signals that come from connections to inputs or ports on the microcontroller. Each port is assigned a specific number that is correlated to the pins on the microcontroller.



1) Analog Inputs and Outputs

Analog inputs, inputs that have a scalar value proportional to some robot condition, allow the ARX microcontroller brain to make decisions about how best to control the robot. On/off digital inputs tell the microcontroller about specific conditions. These inputs can be a constant ON (1) condition or a series of pulses as in the case of IR communication. The IR communications depends upon a series of ON/OFF pulses occurring at a specific frequency. You can think of it like Morse code that the microcontroller interprets to communicate with a remote PC.

Analog **inputs** provide information about the following;

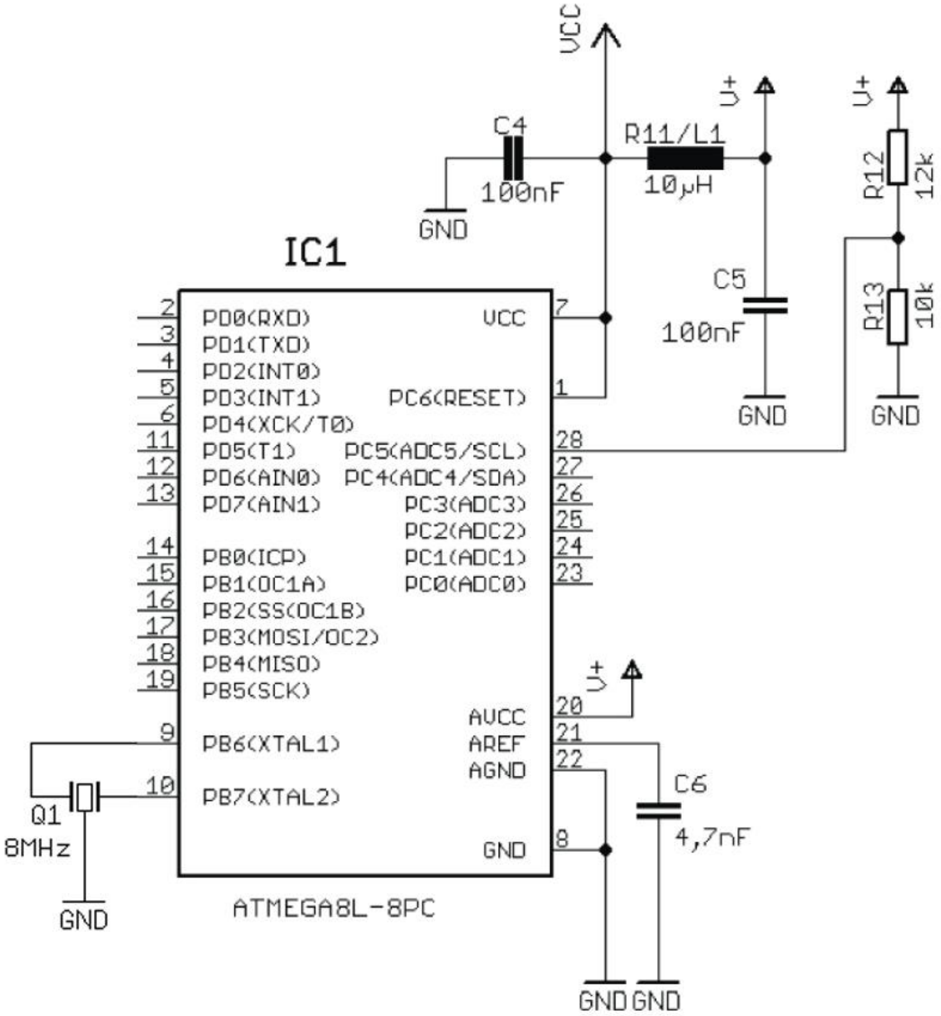
- Battery voltage
- Collision Detection
- Encoder input
- Line Tracing

2) Digital Inputs and Outputs

Digital (on/off) **outputs** are used to control;

- Status LEDs
- Motor Direction
- Digital Pulse outputs control;
- Motor Speed
- IR sending of Data

B. Schematic Diagram



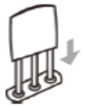
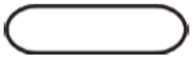
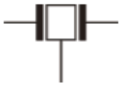
The microcontroller subsystem is the brains- controller for the entire ARX robot. All sensor inputs required to control the robot and all control signals originate at the microcontroller. In addition to these inputs and outputs (which we have covered in the other subsystems), the microcontroller also requires a few additional components and connections that we will review here.

1) The Resonator

First and perhaps most importantly the microcontroller requires a series of pulses that regulate how it executes programs. Some compare it to the heartbeat of the system. These electrical pulses are derived from a resonator, an electrical device that provides pulses at a predefined frequency, in this case 8 megahertz. That is 8 million pulses per second. You can think of these pulses like the ticking of a clock, each tick advances the program being executed by the microcontroller.

In order to synchronize the data transfers for infrared communications and to allow exact timing in programming robot movements, the ARX ASURO requires an accurate clock signal. A crystal element delivers an exact 8MHz clock signal, which is also the internal processor rate for processing commands. ASURO's crystal is a resonator, which is a cheaper element and is working with simpler electronic circuitry. The disadvantage is a slightly reduced tolerance quality (the resonator tolerance is 0.1 %), but this is completely satisfactory for the ARX ASURO applications.

The resonator is Q1 which has three connections: one side is connected to Pin 9 of the Microcontroller, the second side is connected to Pin10. The center resonator connection is to GND, ground.

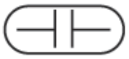
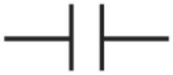
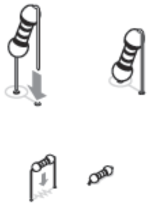
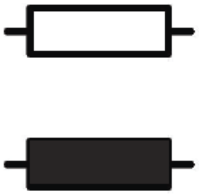
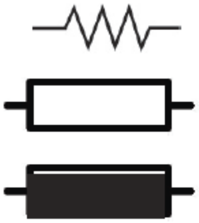
Component	Assembly	PCB symbol	Circuit symbol
Ceramic resonator Q1 	 *No polarity*		

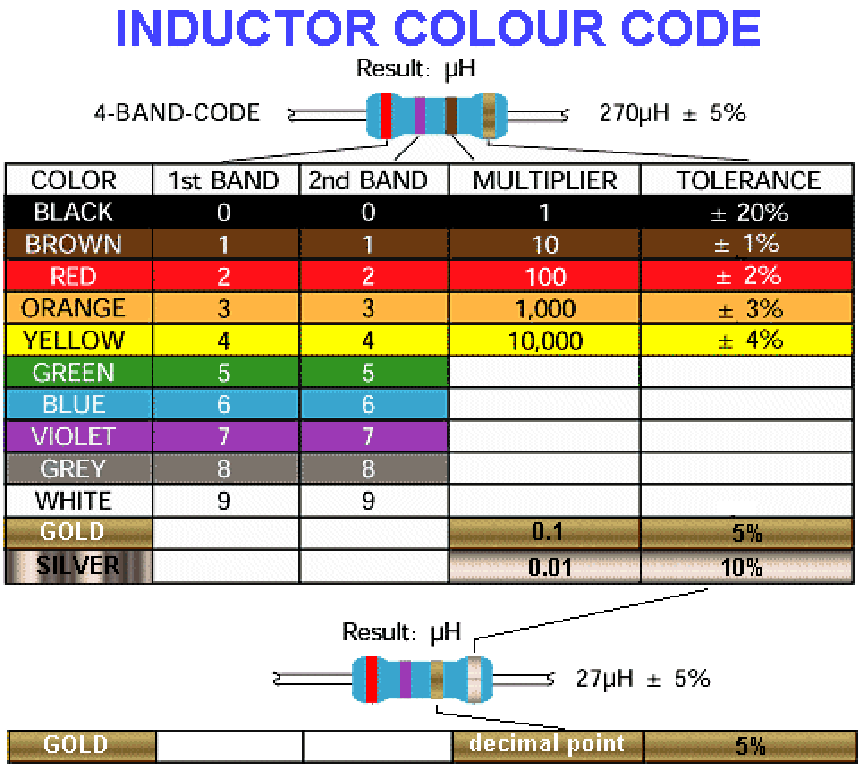
2) Reset Devices

Some microcontroller devices have a Reset switch, that when pressed causes the microcontroller to restart program execution from the very beginning. However our circuit has an automatic reset that is activated whenever power is applied. This consists of a couple of capacitors and an inductor or coil which together form an oscillator. Components for this function are attached to microcontroller Pin1 which is tied to Pin7 and also to Capacitor C4, 100 nanofarads. The other side of capacitor C4 goes to ground.

3) An Inductor – the Coil

Capacitor C4 also connects to Coil R11/L1 which is 10 µH (microhenry). The coil looks like a green colored resistor and also connects to positive power - VCC. The other side of R11/L1 connects to a 100 nanofarads capacitor (C5) and the other side of C5 connects to ground.

Component	Assembly	PCB symbol	Circuit symbol
Ceramic capacitor 			
Resistor/Coil 			


INDUCTOR COLOUR CODE
Result: µH
4-BAND-CODE
270µH ± 5%
Result: µH
27µH ± 5%

COLOR	1st BAND	2nd BAND	MULTIPLIER	TOLERANCE
BLACK	0	0	1	± 20%
BROWN	1	1	10	± 1%
RED	2	2	100	± 2%
ORANGE	3	3	1,000	± 3%
YELLOW	4	4	10,000	± 4%
GREEN	5	5		
BLUE	6	6		
VIOLET	7	7		
GREY	8	8		
WHITE	9	9		
GOLD			0.1	5%
SILVER			0.01	10%

GOLD			decimal point	5%

4) Inductor-Capacitor (LC) Circuit

A capacitor stores energy in the electric field (*E*) between its plates, depending on the voltage across it, and an inductor stores energy in its magnetic field (*B*), depending on the current through it.

If a charged capacitor is connected across an inductor, charge will start to flow through the inductor, building up a magnetic field around it and reducing the voltage on the capacitor. Eventually all the charge on the capacitor will be gone and the voltage across it will reach zero.

However, the current will continue, because inductors resist changes in current. The energy to keep it flowing is extracted from the magnetic field, which will begin to decline. The current will begin to charge the capacitor with a voltage of opposite polarity to its original charge. When the magnetic field is completely dissipated the current will stop and the charge will again be stored in the capacitor, with the opposite polarity as before. Then the cycle will begin again, with the current flowing in the opposite direction through the inductor.

The charge flows back and forth between the plates of the capacitor, through the inductor. The energy oscillates back and forth between the capacitor and the inductor until (if not replenished by power from an external circuit) internal resistance makes the oscillations die out. Its action, known mathematically as a harmonic oscillator, is similar to a pendulum swinging back and forth, or water sloshing back and forth in a tank. For this reason the circuit is also called a tank circuit. The oscillation frequency is determined by the capacitance and inductance values. In typical tuned circuits in electronic equipment the oscillations are very fast, thousands to billions of times per second." (Source: http://en.wikipedia.org/wiki/LC_circuit)

5) What is an Inductor?

An ***inductor****,* also called a coil or reactor, is a passive two-terminal electrical component which resists changes in electric current passing through it. It consists of a conductor such as a wire, usually wound into a coil. When a current flows through it, energy is stored temporarily in a magnetic field in the coil. The unit of measurement for inductance is henry(H) or more commonly the milli-henry(mH) or micro-henry(μH.) *(source:* en.wikipedia.org/wiki/Inductor)

6) What is an Oscillator?

For something to oscillate, energy needs to move back and forth between two forms. For example, in a pendulum on a clock, energy moves between potential energy and kinetic energy. An oscillator can be formed with a capacitor and an inductor. Both capacitors and inductors store energy. A capacitor stores energy in the form of an electrostatic field, while an inductor uses a magnetic field.

The capacitor will start to discharge through the inductor. As it does, the inductor will create a magnetic field. Once the capacitor discharges, the inductor will try to keep the current in the circuit moving, so it will charge up the other plate of the capacitor. Once the inductor's field collapses, the capacitor has been recharged (but with the opposite polarity), so it discharges again through the inductor.

This oscillation will continue until the circuit runs out of energy due to resistance in the wire. It will oscillate at a frequency that depends on the size of the inductor and the capacitor. Oscillators are used in radios, TVs, computers, and many electronic devices. *(source: http://electronics.howstuffworks.com/oscillator2.htm)*

7) Analog to Digital Converter

Pin28 supplies positive power for the Analog to digital convertor circuit and is connected to V+.

8) Power for Analog devices

Pin21 is the reference voltage for analog circuits and is connected to a 4.7 nf capacitor (C6), the other side of which is connected to ground, (GND).

9) Ground Connection for Microcontroller

Pin22 and Pin8 are connected together and go to ground, (GND).

That ends the summary of the circuit connections for the microcontroller subsystem. It is necessary to understand the software programs that are loaded on the microcontroller to fully grasp how the system works. These descriptions are covered in the programming section.

C. Parts Needed

We will build three circuits for this prototype. First we build the power supply circuit. Next we build the microcontroller circuit, and finally we build the status circuit. The status circuit is used to test the operation of the microcontroller circuit.

The parts needed to build the power circuit:

- Breadboard
- Power Supply with AAA batteries
- Push Button Switch
- Capacitor C8
- Short red wire
- Short black wire

The parts needed to build the Microcontroller circuit:

- The breadboard with power circuit
- Power supply with batteries.
- The microcontroller– IC1 (ATMEGA8L- 8PC) (ASURO)
- Resistors – R12 (12 K Ω), (brown, red, black, red, brown)
- Resistor - R13 (10 KΩ) (brown, black, black, red, brown)
- Capacitor – C4 and C5 (100 nF), (104 printed)
- Capacitor - C6 (4.7 nF) (472 printed)
- Coil – L1/ R11 (10 µH) (looks like a green resistor)
- Resonator – Q1 (8 Mhz)(ZTT 8.0 MT printed)
-

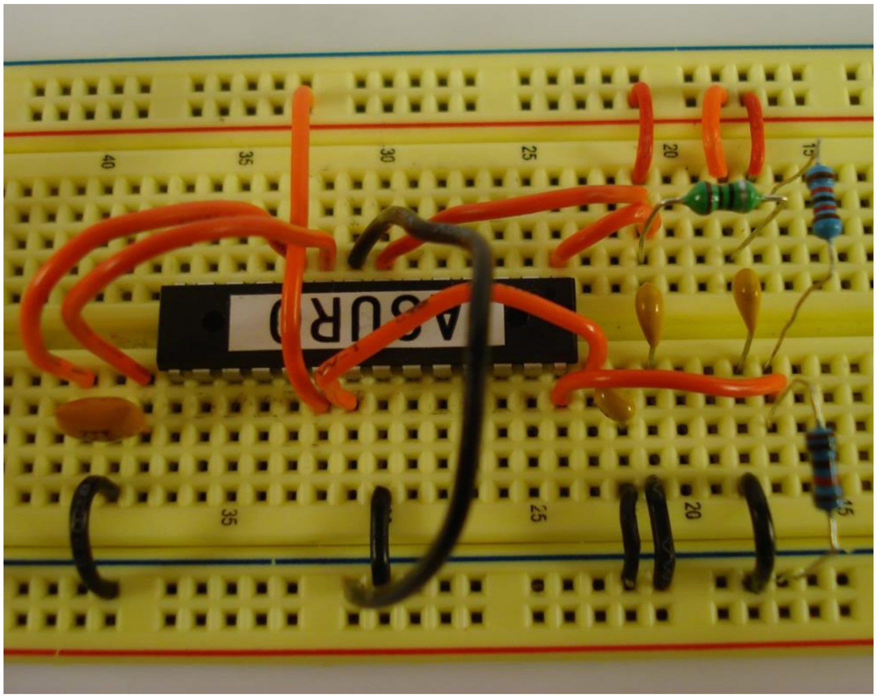
The parts needed to build the status circuit:

- Red LEDS – D11, D15, D16
- Bicolor LED – D12
- Resistors – R19 and R21 (1K Ω) (brown, black, red, gold)
- Resistors R10 and R31 (470 Ω) (yellow, violet, brown, gold)
- Resistor R9 (220 Ω) (red, red, brown, gold)

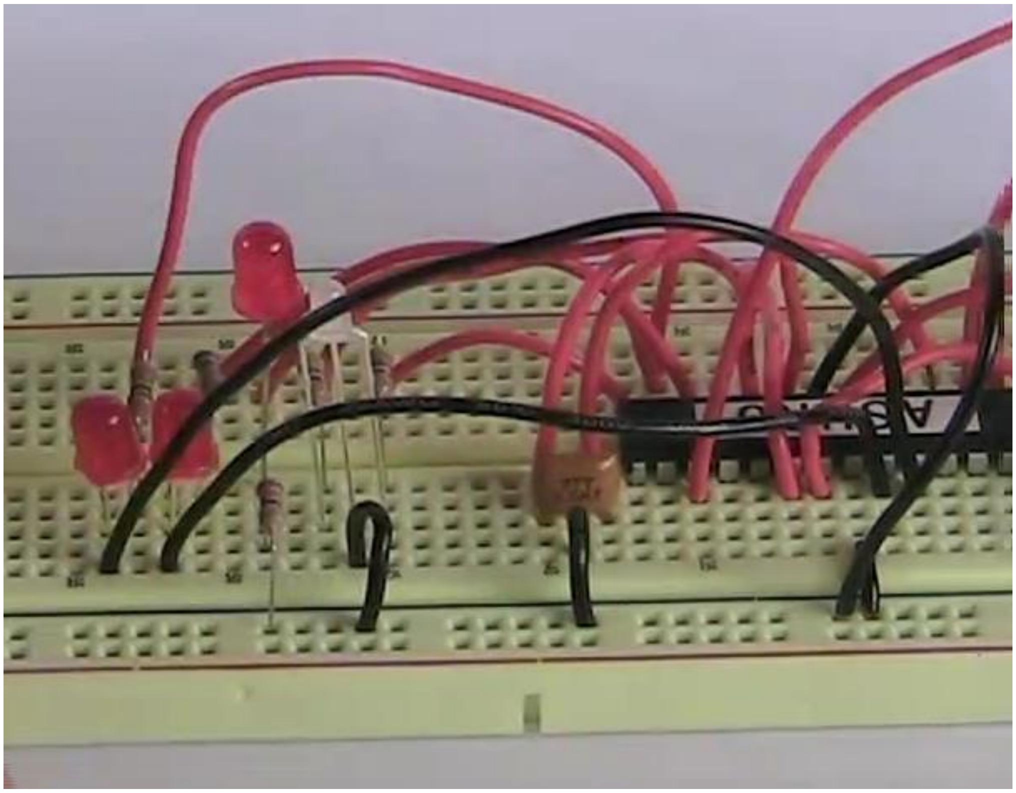
D. Microcontroller Prototype Activity

Assemble and wire the prototype using the schematic diagram, breadboard and parts as illustrated in the Microcontroller Prototype video. Follow the instructions in the Microcontroller Prototype Activity.

Here is an example of a completed Microcontroller Prototype Circuit:



After we build the microcontroller circuit, we also build the status circuit to test it. Here is an example of the completed status circuit:



15. IR COM Link Circuit

How does the ARX ASURO communicate with a PC computer and with humans? Two-way communication with a computer is obtained by using infrared light to transmit information from the ARX to the remote PC. This two way communication allows the ARX to also report information regarding its operation back to the PC.

Because of its widespread use, millions of IR devices are readily available, and IR is a less costly way of communicating with the PC than using a radio type of communication such as BlueTooth or Wifi. We will build a prototype of the IR communication system and use a meter to monitor some of its functions.

A. Overview

The IR Communication subsystem consists of three main elements and some passive components. IR signals sent from an IR equipped PC computer are sensed by a IR semiconductor device.

The pulses, like the on and off of Morse code are transferred to the microcontroller where they are interpreted and acted upon. IR communication can be used to download programs to the microcontroller and then run. The IR can also be used to give specific commands to cause the robot to take pre-defined actions.

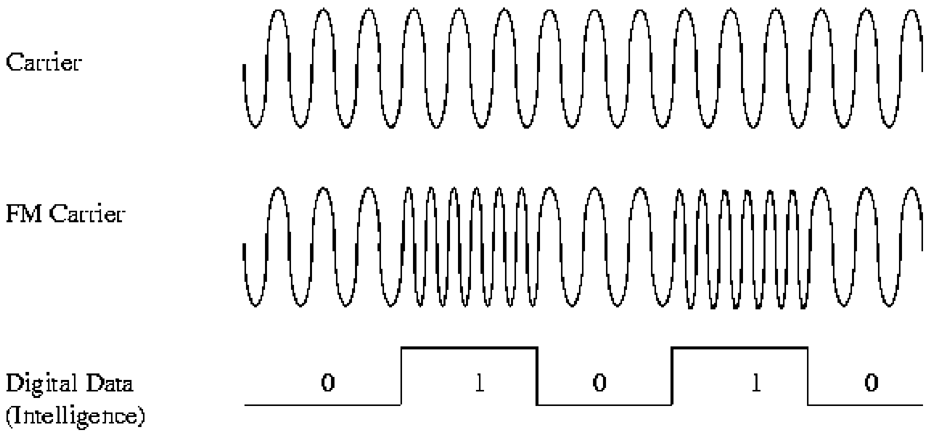
1) IR Signal

IR signals are received at a frequency of 36,000 pulses per second, 36 KHZ. This is what is called the carrier frequency of the device.

The carrier frequency is a frequency capable of being modulated by an information-bearing signal of another frequency. The carrier frequency is referred to as the center frequency in Frequency Modulation (FM), because the carrier frequency is used as a center point, or reference point, with higher and lower frequency signals used to represent 1 bits and 0 bits.

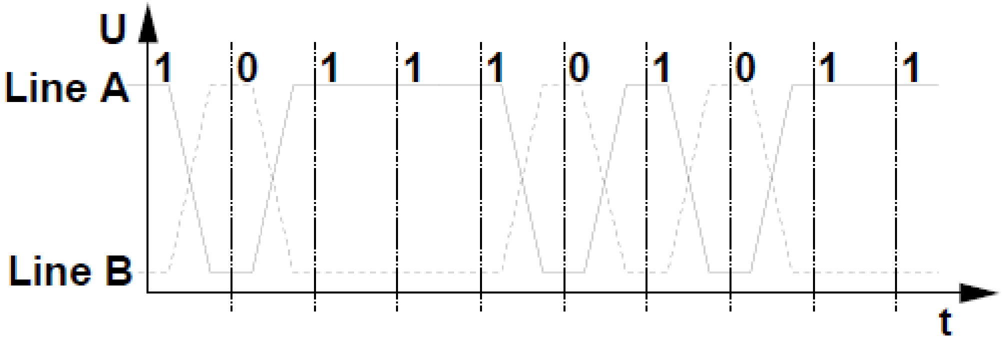
This means that the signals sent between the two devices are sent in a binary code of 1's and 0's. On-off pulses are sent to an IR Diode LED, turning it on and off to provide data transmission. All data is sent in binary streams, which may be in Hexadecimal of 4 bits, 8 bits, 16 bits, or 32 bits, depending on the Integrated Circuits used in the sending and receiving of the data.

The data is converted from characters to binary, then sent as a signal on an infrared light wave, and then converted from binary back to characters when it is received.



2) USB Protocol

The USB uses a differential bus system, indicating two symmetrical lines instead of a single line and a common ground line. A +3.3V level at one line and 0V at the opposite line defines a logical '1'. A logical '0' is defined by 0V5 at one line and +3.3V level at the other).



The USB transfers these signals along two twisted data lines, terminated by a terminating resistor, in order to eliminate reflections.

According to Standard 2.0 the USB-Bus may transfer data at a maximum speed of 480MBit/s (Million Bits/Second), defining a bit interval of 2.08ns (remember 1 second = 1,000,000,000 nanoseconds). The light velocity, also valid for data transfer rates, restricts the physical length of a data bit to a maximum of 625mm[6].

Most USB-cables exceed by far 625mm (one bit length) and we may imagine how several bits in a row are taking place on a cable, while being transferred from transmitter to receiver. They will move along the twisted line pair, terminated by its so-called characteristic line impedance.

What is characteristic impedance? At high frequencies or relatively long transmission lines the repeated transformation from magnetic to electric fields (and vice versa) initiates a characteristic relation between AC-voltages and AC-currents at any point in a cable. The relation depends on the cable structure (isolation sheets, diameter and spacing system) and is called line impedance (impedance is the relation between AC-voltage and AC-current) as a parameter for signal cables transferring AC-signals. But just to be sure: Line impedance is not to be confused with a cable's ohmic resistance, which also depends on cable length. Ohmic resistance is the relation between DC-voltage and DC-current.

In analogy to water, waves reflecting and returning at concrete edge of swimming pool, electromagnetic waves will also reflect at any alterations of line impedances, e.g. at the end of a line. These reflections will cause bits to be returning to the transmitter, where they may be reflected again – and will be mixed with new data bits on their way to the receiver, causing serious data bit congestion at the receiver input terminal!

To avoid chaotic data traffic, the bits will be absorbed into a "dead end alley", a normal resistor valued at the cable's line impedance. The terminating resistor absorbs the electromagnetic energy of the bits and eliminates the reflecting signals. Of course we will need to registrate the input signal before eliminating; otherwise the data transmission would be meaningless.

Normally differential bus systems are quite insensitive to electromagnetic interference, because interferences will symmetrically affect both lines and largely compensate each other.

Consider for instance transmission of a logical '1'. At line A, a voltage 3.3V, and at the opposite line B, a voltage 0V will be applied. An interference signal will increase both voltage levels by - let's say 3V, causing line A to carry 6.3V and line B 3V. Still the voltage at line A is exceeding the voltage at line B and the receiver will be confident in detecting the logical signal '1'. By using symmetrical lines and comparators differential bus systems will still be reliable even in applications using higher data transfer rates.

3) The USB Connector

Analyzing a USB-connector we may easily discover four terminals. Two ports for one direction and the others for the return signal, all differential data lines as described before. Well, this is not quite true! Certainly USB is a duplex-bus-system, in which data may be transferred bi- directionally, but at the same time it is only a half duplex-bus-system, as both directions may only be chosen alternatively and cannot be active in parallel. One pair of data lines may now be used for both directions. The other pair of lines is used as a 5V-power supply for peripheral equipment, which is an extremely convenient way of providing a common power supply for many small units needing small amounts of electrical power. In USB systems the poorly designed power supply by data lines in the RS232-IR-Transceiver may be skipped.

A closer look at the PC-connector reveals another detail. Both inner terminals are located somewhat deeper than the others, allowing "Hot-Plugging" to a running PC-system. Plugging a connector into the PC at first connects to the power supply and after a short delay the data lines will be connected.

4) USB Power and Data Transfer

As already described the USB-Bus provides a 5V-power supply, entering the circuit by ferrite-beads for high frequency interference suppression. C6 suppresses current impulses, caused by pulse modulation of the transmitters infrared diode.

Because the USB-Protocol requires a hub-announcement, a Device-ID transfer, a Device Descriptor, a suggested transfer rate and maximum power current, data format generation, error detecting, providing exact bus clock and data signals we are quite happy to choose an IC (an FT232BM produced by FTDI), to take care of all these difficult jobs and present an output port comparable to a serial interface. The chip also avoids a time-consuming study of several 100 pages of USB-specifications.

A preprogrammed EEPROM will be used to store system parameters. The USB-data signals are transferred by two serial resistors R1 and R2, which together with the receiver input resistor value add up to a line impedance of 100.

Being an asynchronous bus-system, USB does not provide a clock signal. Transmitter and receiver must provide their own clock signals. A resonator Q1, generating a 6MHz frequency is connected to a Phase Locked Loop PLL on chip, multiplying the frequency to 48MHz. This 48MHz clock signal samples the incoming data signal. Our IC will be able to deliver a maximum data rate of 12MBit, so each bit may be sampled four times, improving the signal quality and allowing a synchronization of clock systems. Minor tolerances in free running crystal oscillators require clock synchronization. The integrated on chip oscillator is fed by an RC-lowpass filter (R7, C2) to eliminate a ripple at the main supply voltage system.

A PLL (Phase Locked Loop) is a useful circuit to detect and tune the phase and frequency difference between two signals. We may apply the PLL to design an oscillator based on a low frequency crystal, but generating a few *decades higher frequency. In the first place frequency multipliers consist of a controllable oscillator, generating near the required frequency. Additionally we need an exact oscillator, generating a frequency at an integer fraction N of the required frequency, a frequency divider to reduce the required frequency by the integer fraction N and the PLL to adjust the divided controllable oscillator's output frequency exactly to match the exact oscillator's frequency.*

5) Data Transfer

The ARX sends IR signals using its own IR LED. Similar to the other InfraRed LEDs we have used, this IR LED emits light in the infrared scale. It is used to transmit data from the ARX ASURO to the IR transceiver plugged into the PC. (see photo later in this chapter)

The IR LED, part number SFH415-U is black. From its schematic diagram, we see that the Cathode side is the shorter leg to the flat side of the LED. LED D1 will be emitting light, as soon as the bus transmits or receives data.

Data to be transmitted from the PC to ARX ASURO are transferred by USB into the FT232BM at a rate of 12Mbit/s, where they are stored inside the IC temporarily and then sent slowly bitwise, including start and stop bits, at a rate of 2400Bit/s by TXD-line to the tiny processor IC1 (a preprogrammed ATtiny11-6PC belonging to the same IC-family as the ARX ASURO main processor, but containing fewer peripheral units). This module takes care of the 36kHz modulation, which will be derived from the 6MHz resonator. This generator is accurate enough and does not require a trimmer. Terminal PB0 provides the modulated signal, controlling the output light signal of LED D2. Resistor R10 is a current limiter.

6) The IR Receiver

This module contains a SFH5110 to receive, amplify and demodulate the light signal. The input signal is applied to the RXD-port of the USB-IC, which transfers it by USB to the PC. The supply power of receiver module IC4 is buffered by two capacitors, eliminating the interference.

SFH5110 is an integrated circuit inside a black case with three long pins and a photodiode on the front. A photodiode is a fast, highly linear device that exhibits high quantum efficiency and may be used in a variety of different applications. In this application it is detecting Infrared light. The photodiode detects and receives the IR signal and passes it to the integrated circuit.

The IR Receiver's 1st pin is labeled "out" and it carries the signal of data received to the micro-controller. The 2nd and 3rd pin connect the IR Receiver to Ground and Positive power.

Why are we supplying IC1 from a tiny processor and not directly from 5V? The reason must be found in the USB specification. As soon as the bus is switched into standby mode (e.g., switching the laptop into suspend mode), the supply voltage will remain active, but each bus partner is restricted to a drain current of max. 500_A. As the receiver drains around 1mA, turning the bus into standby mode simply switches off the receiver module completely.

B. Communication Methods

Computers and microcontrollers become more valuable and powerful when they can communicate with other devices. The power of the Internet is an example of the increased value of computers when they communicate with millions of others.

Information is stored on the Internet in computers called servers. These computers have a large amount of memory and hard disk space, communication ability, and special software. Whenever you search or go to a website page you are communicating with the server computers. Their special programs provide the information you are seeking and send it back to your computer. Huge amounts of information (billions and trillions of pages) and much of it is accessible via the Internet. Although it is possible to have private sources of information with limited access on private networks, the internet itself is a network of inter-linked computers. The term network simply means computers that are connected together for some purpose. Computers within a company, a school, a home, or a group of buildings can be connected together with a network. They can share information that is not available to others who are not members of the network.

The way information is transferred from one computer to another defines many characteristics of the network. Information can be transferred by cables or wires, radio devices such as WIFI or bluetooth, or infrared devices. Important characteristics of a network are speed of data transfer, cost, complexity, security, reliability, and ease of maintenance.

1) Infrared Communication

The ARX ASURO uses infrared IR to communicate with a PC. This is a cost effective method of communication. It is also highly secure because it works only on a short range, localized area, unlike radio communications which may be intercepted at a far greater distance. Radio is also more costly, although it has the advantage of being able to transmit and receive beyond a line of sight.

One thing that all of these forms of communication have in common is that they each transmit information thru a series of on-off pulses. The frequency at which the pulses are transmitted changes, but the basic concept is the same. The amount of information transferred per unit time increases as the frequency of the wave increases. An infrared transmission operates at a lower frequency and has a slower information transfer rate than radio (such as WIFI or BlueTooth) or Cable or Phone transmissions rates.

2) Complexity

IR transmission is not complex. It is relatively simple when compared to other forms of communication. The number of components required to make it operate are low. The IR receiving device has only 3 wires or connections: two for power, and the third is the signal or data line. IR transmission consists of simply a resistor and an IR LED connected to a microcontroller. On-off pulses are sent to an IR Diode LEd, turning it on and off to provide data transmission. This means that all data is sent in binary streams, but may be in Hexadecimal of 4 bits, 8 bits, 16 bits, or 32 bits.

3) Reliability

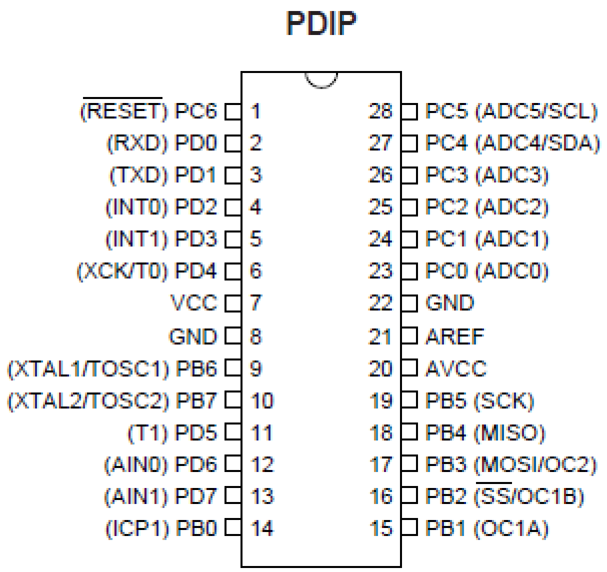
The Infrared communication link is very reliable. The devices involved are straightforward and not very susceptible to failure. Complexity is frequently inversely related to reliability, meaning the more complex a system becomes, the less reliable it is likely to be. Simplicity often yields higher reliability and customer satisfaction. After transmission and reception of the data, devices depend upon a micro-controller program for interpretation of the data. We will discover more about this in the software section.

C. PC Setup for IR Communications

The prototype test requires the installation of HyperTerminal communication software on a PC and setup of the IR device on the com port of the PC. See the separate software installation instructions that came with this package.

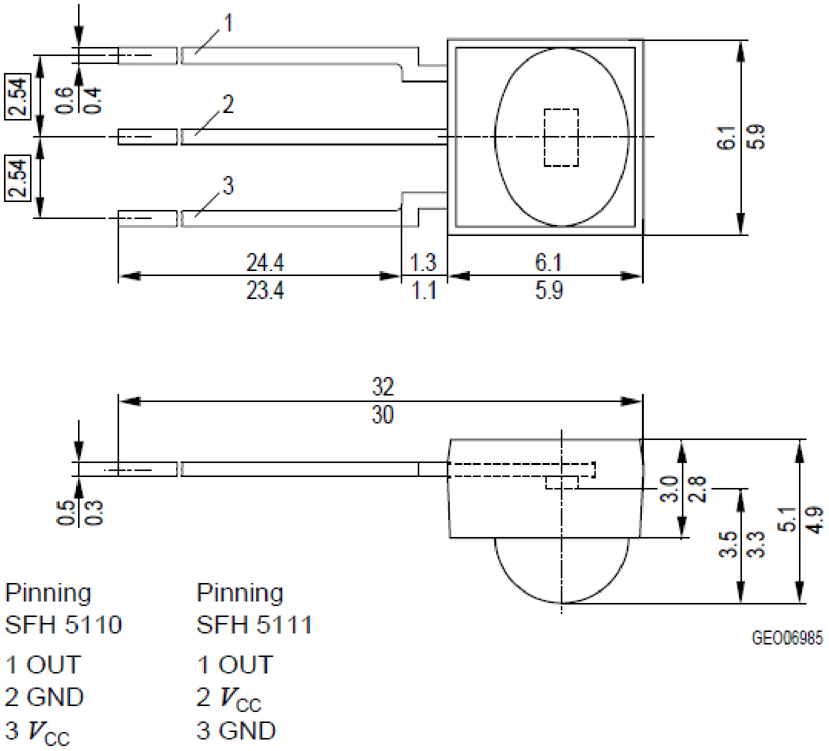
1) Microcontroller IC

The ATmega8 RISC processor is used in this circuit. We will use the breadboard with the microcontroller already wired from the last activity. The pin out diagram below is used to locate the pins for wiring.



2) IR Communication Devices

The IR receiver on the ARX ASURO.

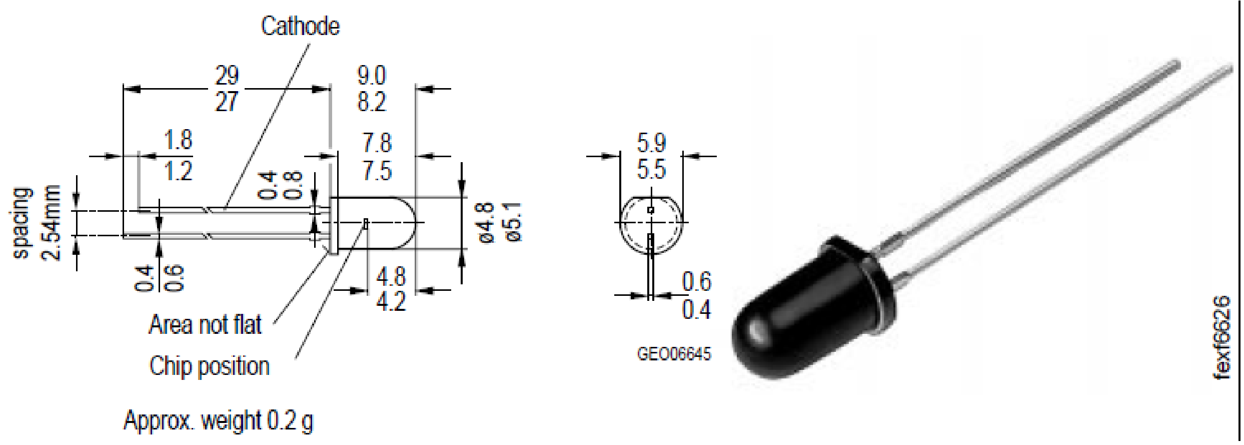


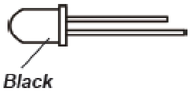
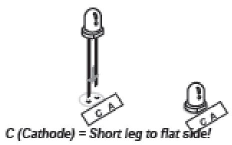
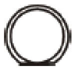
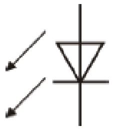
Pinning SFH 5110	Pinning SFH 5111
1 OUT	1 OUT
2 GND	2 V_{CC}
3 V_{CC}	3 GND

Specification of SFH 5110-36

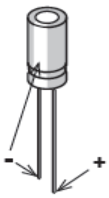
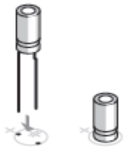
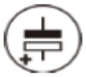
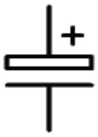
3) IR LED – SFH415-U

Similar to the other InfraRed LEDs we have used, this IR LED emits light in the infrared scale. However, its purpose is quite different. It is used to transmit data to the IR transceiver plugged into PC.

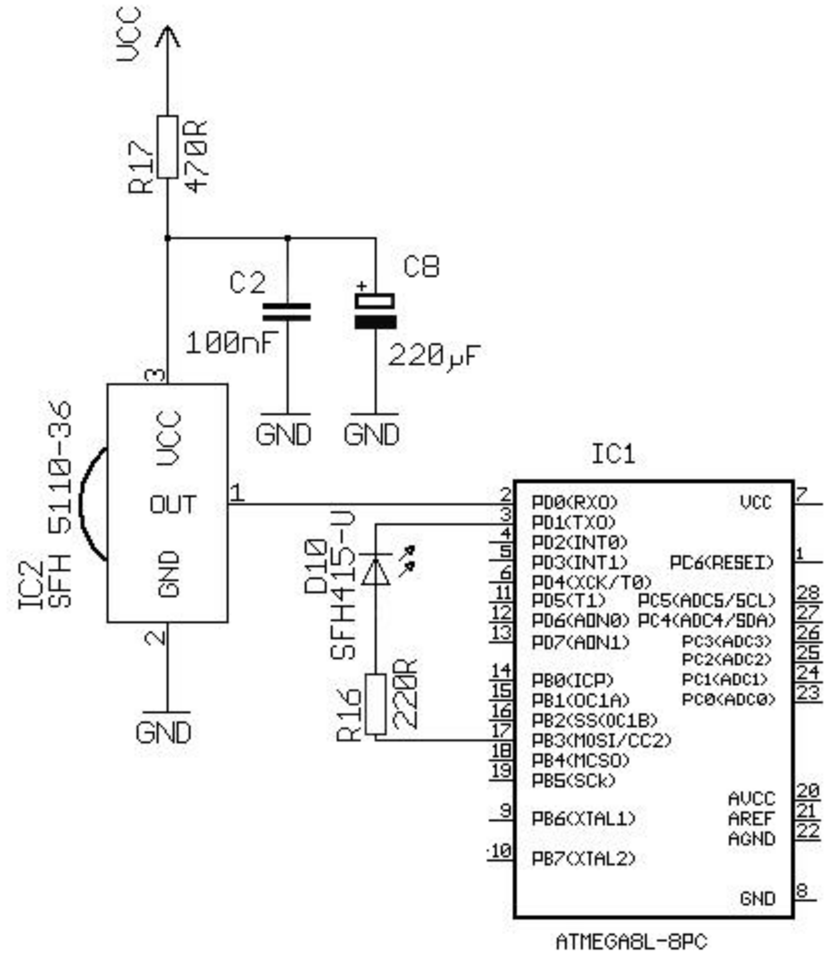


Component	Assembly	PCB symbol	Circuit symbol
IR-LED SFH415-U 			

4) Capacitor C8 (ELCO)

Component	Assembly	PCB symbol	Circuit symbol
Electrolytic capacitor (ELCO) 			

D. Schematic Diagram



As seen in the schematic diagram, the infrared communication subsystem is a relatively simple circuit. It consists of one 470 ohm resistor (R17), two capacitors: C2 is 100 nano-farads and C8 is 220 micro farads, an IR sensing Integrated Circuit, IC2 (SFH 5110-36), which is monitored by the microcontroller IC1,(ATMEGA8L-8PC).
Positive power is connected to Pin 3 of the IR IC through resistor R17. The two capacitors that also connect to Pin 3 of the IR IC filter unwanted electrical noise on the power, and are connected to Ground. Pin 2 of the IR IC is also connected to Ground.

Pin 1 on the IR IC2 connects the IR IC to the microcontroller IC1 via its Pin 2. The microcontroller monitors the electrical pulses from the IR IC and interprets the data and converts it into either a program which is being downloaded or a message from a remote PC which contains commands directing the robot's action. We will later see how the IR communication program performs these tasks in the software section.

E. Parts Needed

Parts needed to build the IR COM Circuit Prototype:

- Breadboard with Microcontroller and Power circuits

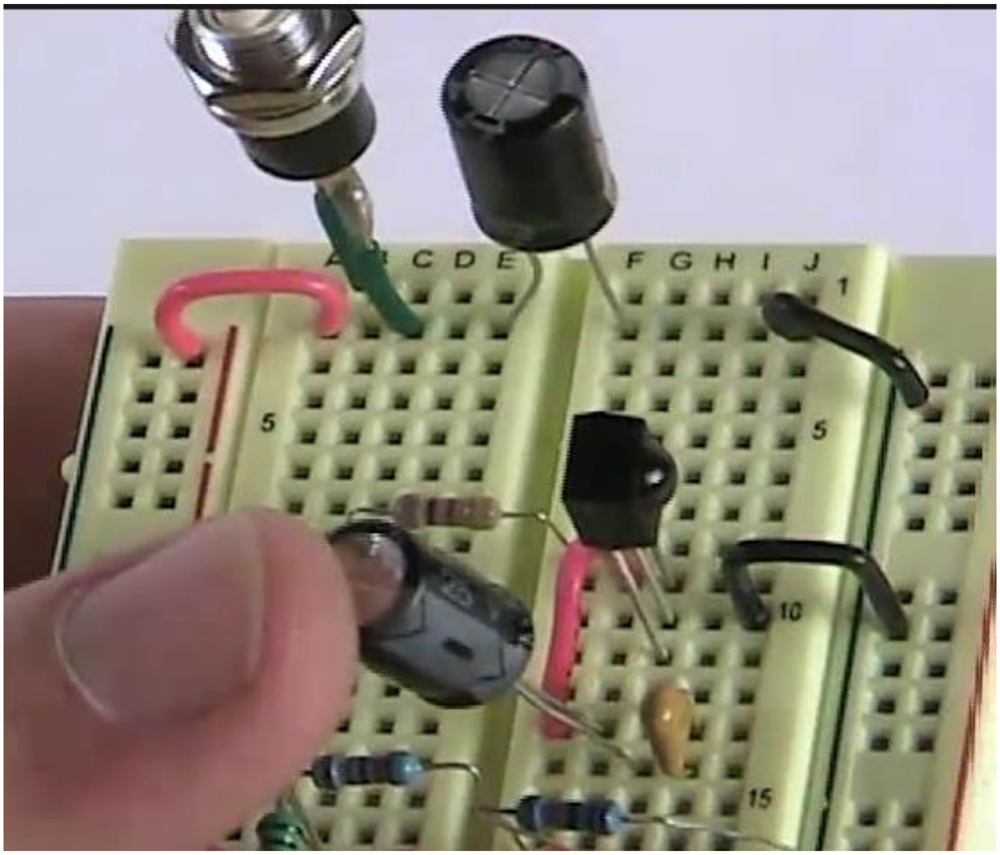
- Power Supply with batteries
- USB IR communication dongle
- Resistor (R17) – 470 Ω (yellow, violet, brown, gold)
- Resistor (R16) – 220 Ω (red, red, brown, gold)
- Capacitor (C2) – 100nF ceramic capacitor, imprint: 104
- Capacitor (C8) – Elco 220µF 10V
- IR-Diode (D10) (black) Pay attention to polarity!

F. IR Com Link Prototype Activity

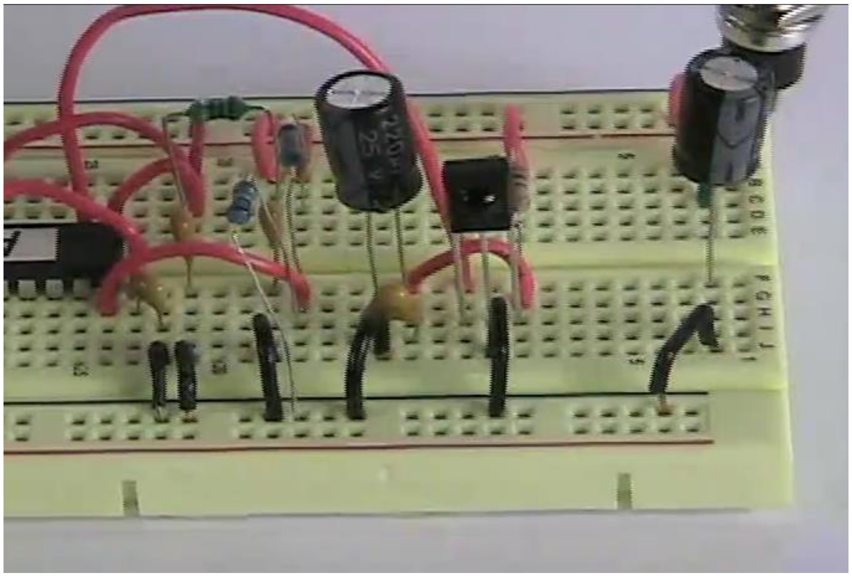
Assemble and wire the prototype using the schematic diagram, breadboard and parts as illustrated in the IR COM Prototype video. Follow the instructions in the IR COM Link Activity.

This circuit also requires the Power supply circuit, microcontroller circuit and status circuit from the last activity.

CAUTIONS:
The IR-receiver IC2 SFH5110-36 is sensitive to ESD (electrostatic damage). It may be damaged by simply touching or even reaching for these parts if you have been charged with electricity (eg. by walking on a carpet). Before handling these parts you should discharge yourself with a discharging bracelet, connected to ground or at least by touching a grounded heater system or a metallic case of equipment.



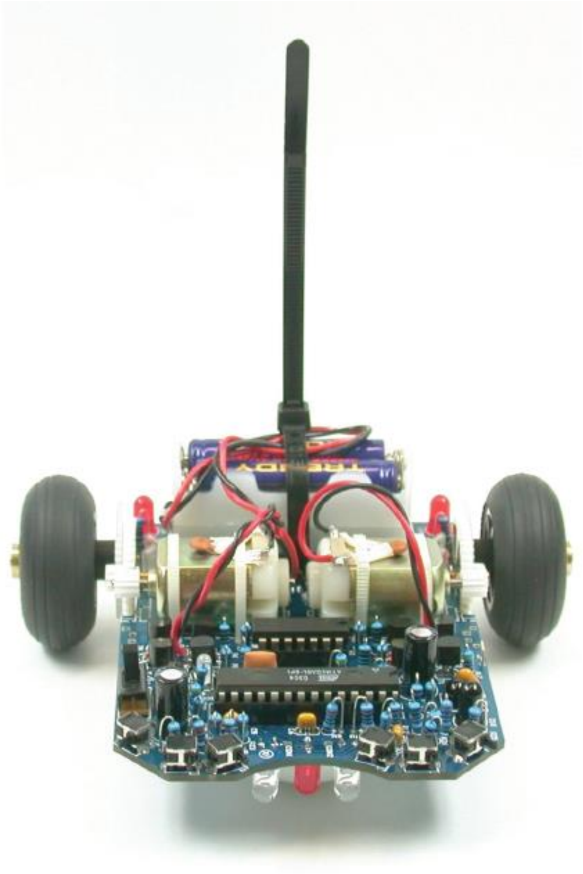
Capacitors in parallel



Connecting from pin 1 of the IR IC to pin 2 of the microcontroller IC

SECTION 3: PC Board Robot Assembly

- *Overview*
- *Assemble Axles and Chip Sockets*
- *Assemble Power Supply Circuit*
- *Assemble Status Circuit*
- *Assemble Collision Detection Circuit*
- *Assemble Motor Control Circuit*
- *Assemble Encoder Circuit*
- *Assemble Line Follower Circuit*
- *Assemble IR Com Link Circuit*
- *Assemble Microcontroller Circuit*
- *Assemble Motors and Wheels*
- *Install Chips and Test Operation of the ARX ASURO*
- *Attach Ball and See Robot Run*



Section 3 Overview

OK the time has arrived, now we move into the fun part! In this section we will be assembling the robot using a printed circuit board (or PCB). All the robot parts - mechanical, electrical and electronic - are attached or soldered to one PCB. This makes a highly efficient robot that is also very rugged and light.

We will be assembling, wiring, and soldering components onto the PCB following a sequence similar to what we followed in building prototypes in Section 2. As you have completed the previous section, you are already familiar with each subsystem and how they operate as well as the components that we use in each. So we will assume that you can identify each component and know how it is oriented for polarity.

We will also be using the circuit diagrams to guide our assembly. But unlike the breadboard, we will not be adding wires to connect components together. Those connections are already made on the PCB board.

Before we begin the assembly process, let's review again how the engineers designed the robot. It will be good to keep these in mind as we put the parts together.

1) **Design Objectives**
   Some of the design objectives for the ARX ASURO were to have a robot which is highly responsive, accelerates, and decelerates rapidly, turns quickly, and has a high top speed. They also wanted to keep the costs low for educational use. Some of the design choices that were made and the ways used to accomplish this are:

   - Keep weight of robot low by not adding additional parts and incorporate the chassis into the PCB board.
   - Use a small high speed DC motor that is lightweight
   - Reduce speed using gears and also increase torque (but still have a relatively high rpm rate on motors even after reducing with gears).
   - Minimize number of electronic components needed for control and thus reduce size of PCB board and weight of robot.
   - Use size AAA batteries which are lighter and are smaller while still maintaining a good amount of run time.
   - Use alternative combinations of hardware and software to cleverly achieve control objectives which minimize parts and minimize I/O requirements of microcontroller. Examples are in the motor control H bridge circuit and collision detection circuit.
   - Make it easy to navigate and follow a preprogrammed path. This is met with using an optical encoder system to facilitate navigation and speed control by incorporating a simple optical encoder mounted on the primary drive gear.
   - Provide a method of simple wireless communication between the robot and a controlling PC. Also allow for downloading programs

to the robot. To reduce costs and maintain simplicity in design this uses an infrared based system.
- Provide a visual method of indicating proper operation of systems. Lightweight and low cost LEDs are used as status indicators.
- Provide a method of line following that is lightweight and simple to implement. Method chosen is based on infrared LEDs in combination with phototransistors.
- The robot needs to have good traction, good grip, and respond well to rapid acceleration and deceleration. The wheels and tires are made of rubber and are sized to increase speed at a high motor rpm.

2) Keys to Success

Keys to success in this assembly process are:

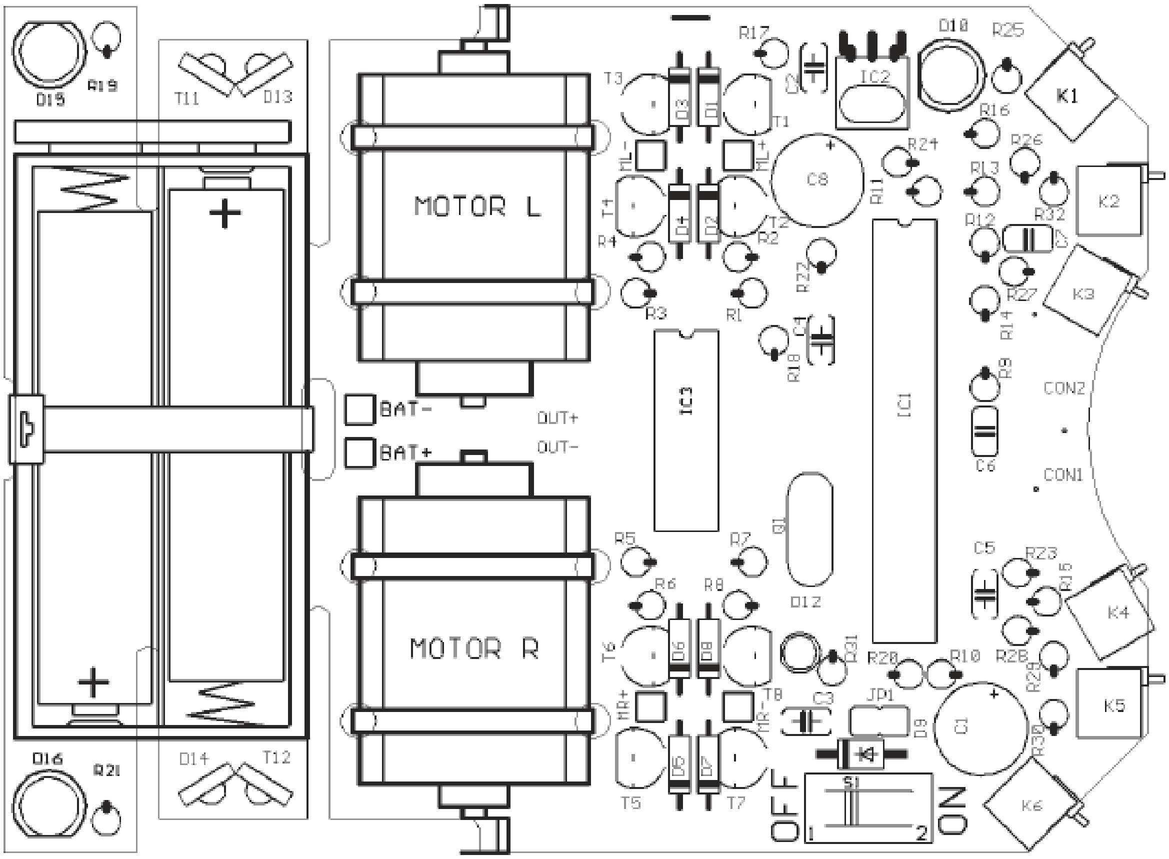
- Take your time and be careful at each step.
- Be sure that each component you insert is the right one.
- Be careful soldering. Be sure to create a good solder joint without any bridging of solder to adjacent areas.
- Double check your work before and after soldering each circuit.
- Don't rush. This is not a race.
- Mistakes are easily made and difficult to correct when soldering. So it is far better to take a little longer and avoid making them in the first place!
- Before soldering the PCB, it is good to practice on a much smaller board and with other devices.

If you did not go thru the lesson on soldering in the Overview section, now is a good time to go back and do that activity. Or if it has been a while, it is good to go back for a quick review and practice the soldering techniques.

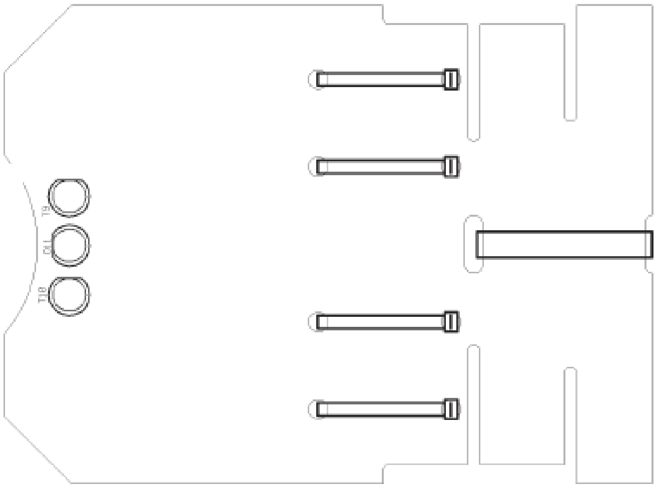
3) Teamwork

If possible, work in a team, with each person taking responsibility for identifying the parts, defining the proper orientation of the parts, soldering, and checking the solders.

- This is an excellent time to use team work to cross check each other's work.
- Check each step carefully. Follow along with the diagrams and instructions. Remember it is as much the responsibility of the checker to avoid errors as it is the person doing the original work.
- Whether you decide each person has a single role, or everyone takes a turn at each role, each person takes responsibility for the final outcome and can take pride in the work of the team.
- This is great practice for working in teams on a job, and using quality control measures to do a good job.
- Remember that everyone makes mistakes. That is why there is an eraser on the end of each pencil!
- Tell your team partner when you find an error without being critical, or gloating. You are part of a team and each team member will make errors. Find and correct errors, move on, and be grateful for your partner.



TOP OF PCB BOARD Showing Component Placement



BOTTOM OF PCB BOARD Showing Component Placement

16. Assemble Axles and Chip Sockets

Before we begin to add electronic components to the PCB board we insert and solder the axles which will support the gears and wheels. We do not add the gears or wheels in this step, they would be in the way while we added the electronic parts for the subassemblies.

A. Assemble Axles

We begin by attaching the axles and chip sockets to the Printed Circuit Board. The axles are soldered on first. The two shorter axles are attached on the upper side of the PCB near the middle of the board. The two longer axles are attached on the bottom side near the back end of the PCB board.

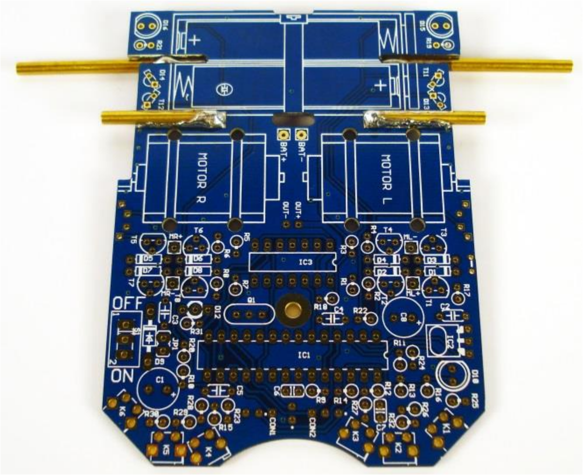
Sockets are devices that hold and connect other electronics into PCBs. Components are usually soldered permanently into PCBs. Once soldered, it is laborious to unsolder and remove the components. For this reason we use sockets which can be soldered into the PCB to hold other components such as integrated circuits. This allows us to easily remove an integrated circuit from the PCB and also reduces the chance of overheating the IC during soldering.

The smaller socket with14 legs or pins is for the Quad AND Gate Integrated circuit, (Quad meaning 4). This device is used in the motor control circuit.
The larger socket with 28 pins is used for the microcontroller integrated circuit. Both sockets consist of a bottom area with protruding pins, that go thru the PCB and are soldered into the PCB on the back, and an upper area with holes that allow the IC chip to be plugged in.

We will not insert the IC chips at this point, we will just solder the sockets into place. The ICs are very susceptible to heat and to electrostatic charge, so we will save that step for later.

1) Parts Needed

- PCB Board
- Two short axles which are 24.5 mm
- Two long axles which are 42 mm long



2) Axles Assembly Process

The axles are manufactured from a brass rod. A pair of axles (which are 24.5 mm and 42 mm long) are needed for each motor located on either side of the chassis. The two shorter axles are attached on the upper side of the PCB on the left and right near the middle of the board. The two longer axles are attached on the bottom side on the left and right near the end of the PCB board.

Clean the axles before attachment with some fine sandpaper (240 or more grit) to improve adhesion of the solder or glue. The axles may be soldered on or glued. Soldering is the best option because it hardens much faster than glue and can be corrected by de-soldering. (see video)

We begin with one of the short axles. Put the PCB on a table with the bottom side facing up. Push the longer axle fully to the end of the slide. The axle should be lying flat.

First Tin the axle as shown in the soldering video. Then wet the soldering iron tip with some solder and press the tip onto the axle. After heating the axle add solder at the soldering pads beside the axles.

After the soldering is finished, remove the soldering iron while pressing the axle downwards with a screwdriver until the soldering connection is cooled off completely.

Repeat this procedure with the other three axles. Let the axles cool down before touching. Any soldering fractions on the surface of the axles in the area where the wheels will be attached can be removed with sandpaper or a fine file.

B. Assemble Chip Sockets

There are two integrated circuits on the ARX ASURO – the microcontroller and a logic gate. These are not inserted directly onto the Printed Circuit Board, but instead are inserted into a chip socket. We need to install those chip sockets and solder them to the PCB board.

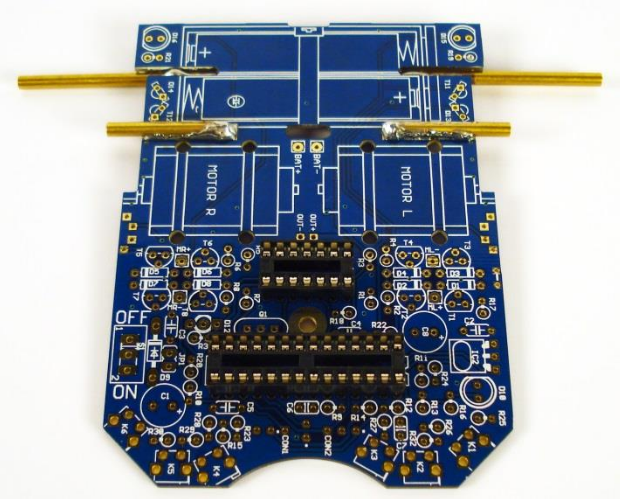
Sockets are devices that hold and connect other electronics into PCBs. Components are usually soldered permanently into PCBs. Once soldered, it is laborious to unsolder and remove the components. For this reason we use sockets which can be soldered into the PCB to hold other components such as integrated circuits. This allows us to easily remove an integrated circuit from the PCB and also reduces the chance of overheating the IC during soldering.

The smaller socket with14 legs or pins is for the Quad AND Gate Integrated circuit, (Quad meaning 4). This device is used in the motor control circuit.
The larger socket with 28 pins is used for the microcontroller integrated circuit. Both sockets consist of a bottom area with protruding pins, that go thru the PCB and are soldered into the PCB on the back, and an upper area with holes that allow the IC chip to be plugged in.

We will not insert the chips into the sockets at this point. They are very susceptible to heat and to electrostatic charge, so we will save that step for later.

3) Parts Needed

- PCB Board from prior step
- IC1: either one Dual in line 28 pole-socket or two Dual in line 14 pole-sockets
- IC3: Dual in line 14 pole-socket



4) Chip Socket Assembly Process

Assemble IC chip sockets as shown in Axle and Chip Socket video

- Place chip sockets on board with wires protruding through holes
- IC1: **insert only the socket** (either one Dual in line 28 pole-socket or two Dual in line 14 pole-sockets). Pay attention to polarity! A slight asymmetry is a mark for polarity in the socket and in the symbols on the PCB!

- IC3: **insert only the socket** (Dual in line 14 pole-socket). Pay attention to polarity! A slight asymmetry is a mark for polarity in the socket and in the symbols on the PCB!
- Turn board over and solder the wires for each socket

17. Assemble Power Supply Circuit

All electronic devices require a power supply of some type.   The ARX ASURO uses a battery pack as its power source. In addition to a power source, there is usually a method of turning the power on and off - most often this is a switch.

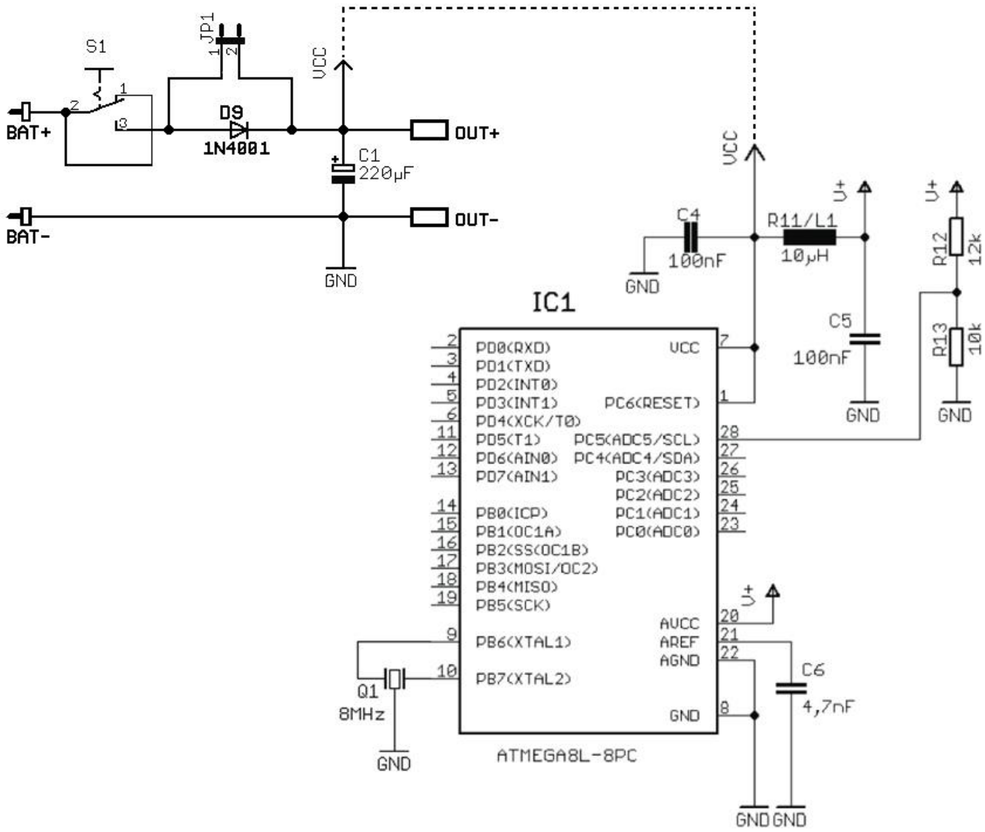
Most power supplies may also have a method of conditioning or regulating power. For the ARX ASURO, there is a diode which is used to reduce voltage when using non-rechargeable batteries. There is also a capacitor to assist in regulating the power.

When building the PCB power supply there is a jumper connection that bypasses the diode for use with rechargeable batteries. If using non-rechargeable, the jumper connection is removed so the diode will reduce the voltage to an acceptable level.

Care must be taken when mounting and soldering the components. Take your time and follow the video and assembly instructions. They illustrate each step of the assembly process.

Note: The power supply case with batteries is not attached to the PC board during this step. It is susceptible to excess heat from soldering and will get in the way of soldering other components. We will add it in a later step.

A. Schematic Diagram



A 4 cell AAA battery pack provides power for the ARX robot. The positive conductor of the battery pack connects to Pin 2 of switch S1 - the wiper of a sliding On/Off. The normally open output of S1, (Pin3) is connected to jumper JP1 and to the anode of diode D9.

Two types of batteries may be used with the battery pack, rechargeable and Non-rechargeable. Non-rechargeable batteries output a higher voltage approximately 6 volts, than rechargeable batteries at about 4.8 volts. This is because a rechargeable battery outputs about 1.2 volts and a non-rechargeable battery outputs about 1.5 volts.

When batteries are connected in series the plus of the first is connected to the negative of the second and the positive of the second is connected to the negative of the third and the positive of the third is connected to the negative of the fourth as they are in the battery pack then the voltages add together. Thus the total voltage is the sum of the individual voltages of each battery about 6 volts for non-rechargeable and 4.8 volts for rechargeable.

Our circuits are designed to operate at no more than 5.5 volts so we use the Diode D9 to drop the voltage when using non-rechargeable batteries. Diode D9 will drop the voltage by about 1 volt. When rechargeable batteries are used the total voltage does not need to be dropped thus jumper JP1 is inserted to bypass D9.

The power supply output is connected to C1, a 220 micro farad capacitor which helps stabilize the output voltage. The positive side of the power supply is connected to VCC and OUT+, and the negative to GND and OUT-.

Power supply voltage is monitored by the microcontroller at Pin 28 which is analog to digital converter ADC5, that is connected to a voltage divider consisting of R12 12K ohms connected from VCC to R13 10K ohms and then to ground.

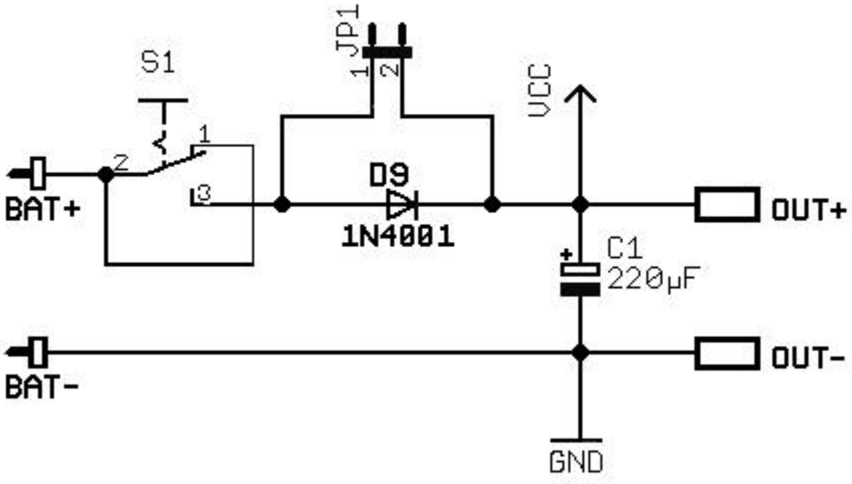
These two resistors R12 and R13 divide the power supply voltage by the ratio of their resistances thus the input at Pin28 will be equal to 10,000 divided by 12000 (10K divided by 12K) or 5/6 of the full voltage.

B. Parts Needed

- PCB board from prior assembly
- Diodes – D9
- Capacitors – C1 (220 µF)
- Switch – S1
- Jumper wire – JP1
- Resistors (Voltage Monitoring) – R12, R13

C. Procedure

To build our Prototype, we will use this partial diagram:

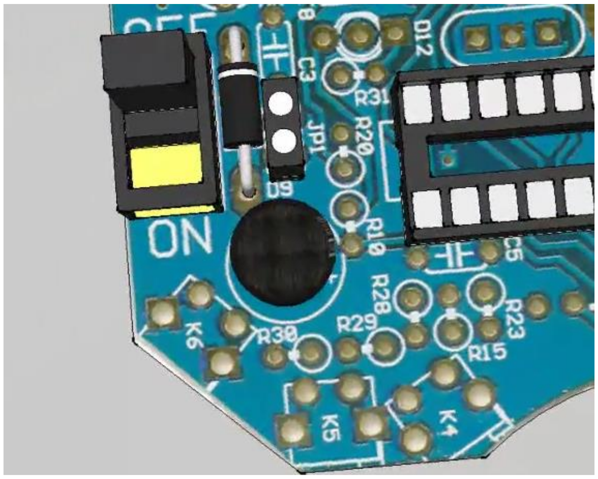


Assemble and wire the power supply using the PCB board and parts following the schematic diagram and steps illustrated in the Power Supply PCB Assembly Video.

1. Locate the components and place them on top of a printed circuit diagram. Look up the polarity of items and note them on the diagram.
2. Solder the components in this order:

- Jumper JP1
    - Locate J1's position, insert it, solder its leads, if trim is required trim excess lead length.
- Diode D9
    - Locate D9's position, bend D9's leads for horizontal mounting, insert D9, solder its leads, and trim excess lead length.
- Switch S1
    - Locate S1's position, insert it, and solder its leads.
- Capacitor C1
    - Locate C1's position, note polarity, insert it, solder its leads, and trim excess lead length.

3. Test the circuit board using the Multimeter
4. Record test results in worksheet.
5. Compare test results to prototype test results



3D illustration of completed power supply circuit assembly

18. Assemble Status Circuit

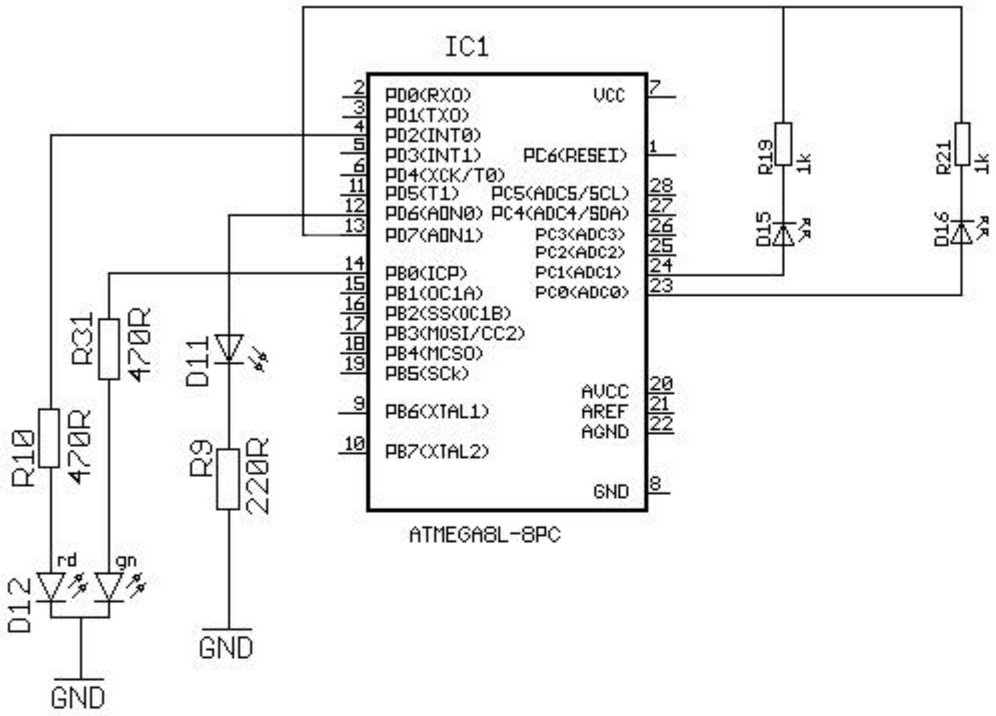
The status circuit uses LEDs combined with current limiting resistors to indicate various robot status conditions. The LEDs are located at various places on the board. One status LED is a bicolor LED with green and red combined. This LED can also produce other color combinations by turning the red and green on concurrently which produces an orange color.

This bicolor LED has three leads, one is a common ground cathode and the other two are the positive or anode for the red and green LEDs. It is the primary status indicator and is used to inform the user of several operating conditions. The three leads (all different lengths) are placed into adjacent holes, (shortest is the Green side of the LED, mid length is the Red side, and the longest is the ground).

It is important to get the orientation and polarity correct on all the LEDs. Before soldering, double check each LED for polarity. Also be sure that the current limiting resistors are the correct value by rechecking their colors against the color chart.

The resistors are in series with either the anode or cathode side of the LED. It limits current to the LED to protect it from burnout and reduces its power consumption. Review the schematic diagram, the parts list, and follow the video and assembly instructions which illustrate each step of the assembly process.

A. Schematic Diagram



(Note: circuit with R9 and D11 are part of line following and will be installed later)

B. Parts Needed

- PCB board from prior assembly
- Top LED = D12 (bicolor – red/green)
- Back LEDs = D13 and D14 (red)
- Resistor R10, R31, R22 = 470 Ohm (yellow, violet, brown, gold)

C. Procedure

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Status Circuit Assembly Video.

Insert components into PCB and solder the components in this order:

1. LED D12 (Bicolor)
2. Resistor R31: 470Ω (yellow, violet, brown, gold)
3. Resistor R10: 470Ω (yellow, violet, brown, gold)
4. LED D15 (5 mm red). Pay attention to polarity (short leg must be inserted at the mark)
5. Resistor R19: 1KΩ (brown, black, red, gold)
6. LED D16 (5 mm red). Pay attention to polarity (short leg must be inserted at the mark)
7. Resistor R21: 1KΩ (brown, black, red, gold)

Test the circuit using the Multimeter
Record test results in worksheet
Compare test results to prototype test results

D. Assembly Steps

Note that we will first show the device or component we are going to install on the schematic and then proceed to show the actual installation. Note that each component has a printed part or component number adjacent to its location on the PCB. Also note the orientation of the LED s it is being inserted.

Installing D12 Bicolored LED
We will begin by installing D12, the bicolored LED as indicated in the schematic. We show the component on the 3D CAD illustration. Next we show D12 being inserted into the PCB. (Shortest leg is the Green side of the LED, mid length is the Red side, and the longest is the ground). The shortest leg goes toward the text D12 on the board. Notice the flat side of the LED is marked on the board to help orient it.
Next we turn the PCB over and prepare to solder the three LED legs by first bending one or two of them to keep it in place. Next proceed to solder each leg as shown in the video. Then trim off the excess length of legs.

Installing R31, 470 ohm Resistor
Next we will install and solder the green LED segment current limiting resistor R31 470 ohms. First we bend the Resistor leads for vertical installation. Identify the

location for R31, adjacent to D12 insert the resistor, turn the PCB over and solder the resistor leads.

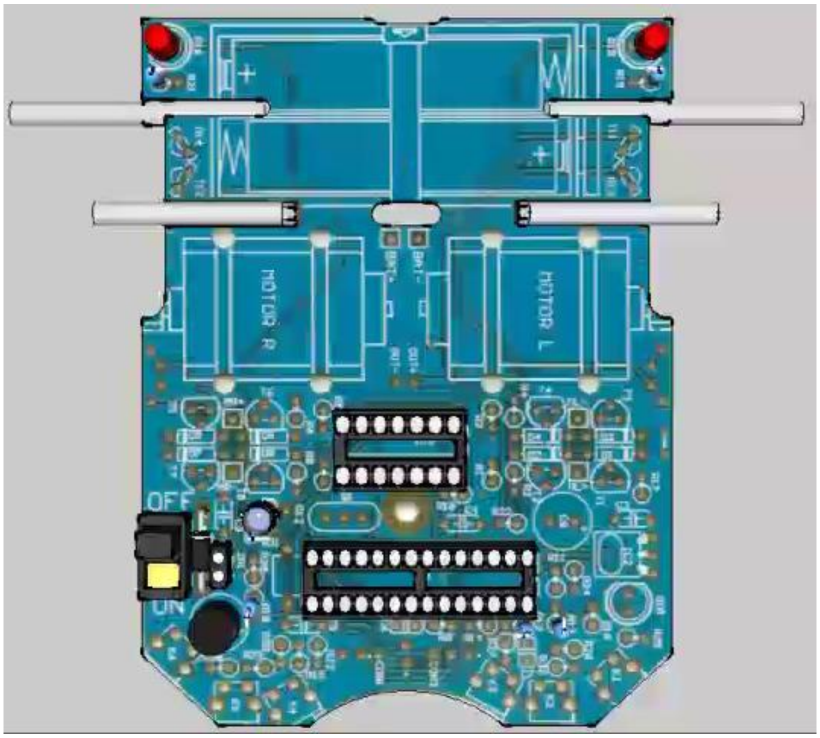
Installing R10, 470 ohm Resistor
Next we will install and solder the red LED segment current limiting resistor R10 470 ohms. First we bend the Resistor leads for vertical installation. Identify the location for R10, adjacent to R31 insert the resistor, turn the PCB over and solder the resistor leads.

Installing D15 Red LED
As you install D15 be sure to observe the orientation of the LED: note that the flat side of the LED is shown on the printed component location on the PCB. After inserting the LED turn the board over and proceed to solder the LED leads.

Installing R19, 1K ohm
Locate the position for R19 adjacent to D15, Bend R19's leads for vertical insertion. Insert R19 turn the PCB over and solder the resistor leads.

Installing D16 Red LED
As you install D16 be sure to observe the orientation of the LED: note that the flat side of the LED is shown on the printed component location on the PCB. After inserting the LED turn the board over and proceed to solder the LED leads.

Installing R21, 1K ohm
Locate the position for R21 adjacent to D16, Bend R21's leads for vertical insertion. Insert R21 turn the PCB over and solder the resistor leads.



19. Assemble Collision Detection Circuit

The collision detection circuit includes six switches on the front of the robot. Each switch is accompanied by one or more resistors.

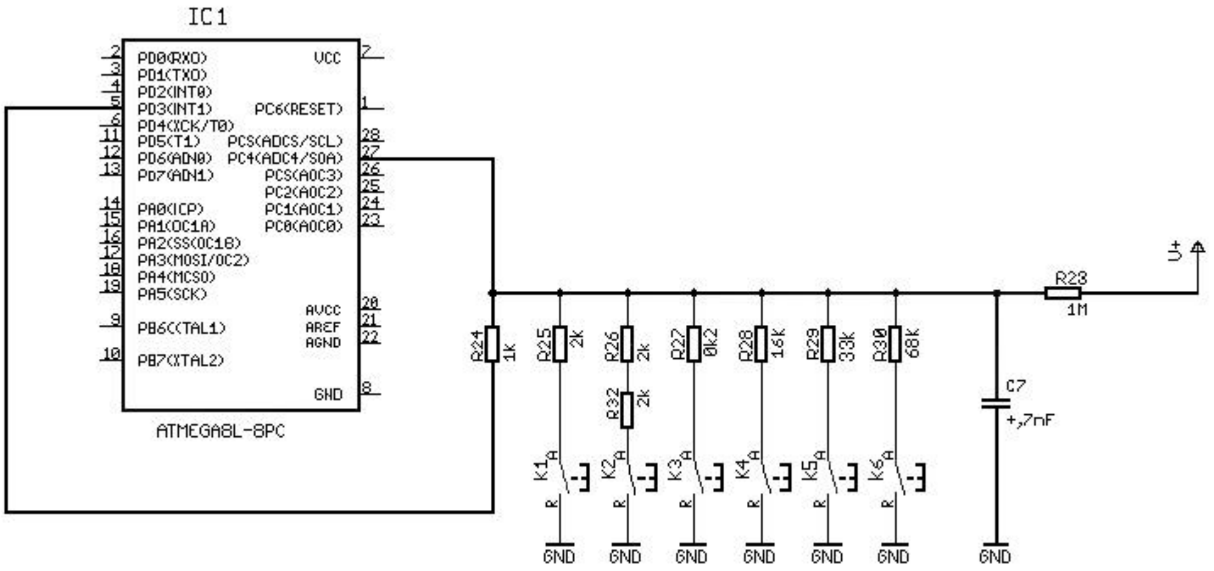
When assembling the collision detection circuit it is important to observe which resistors are used with which switch. As you recall from building the prototype, the resistor values allow the microcontroller to determine which switch has been pressed. Use the resistor chart to double check the color codes for each prior to soldering.

As always caution and care in soldering is important to avoid solder bridges and poor solder joints. The switches need to be oriented as indicated on the PCB board.

A. Schematic Diagram

The circuit diagram for the collision detection system is shown below. This is one part of the overall diagram for the ARX ASURO printed circuit board. Notice that it only has one line going to the input of the microcontroller, and one line going to the A/D converter. The designers have figured out a way to send a signal to the microcontroller which tells it which switch (or combination of switches) has been depressed using only one input.

There is one capacitor, five switches, and seven resistors in this circuit.



B. Parts Needed

- PCB board from prior assembly
- Switches – K1,K2,K3, K4, K5, K6 (must be mounted flat to the PCB surface!)
- Resistors –
  - R23 (1M), (brown, black, green, gold)
  - R24(1K), (brown, black, black, brown, brown)

  - R25(2K), R26(2K), R32(2K), (red, black, black, brown, brown)
  - R27(8.2K), (grey, red, black, brown, brown)
  - R28(16K), (brown, blue, black, red, brown)
  - R29(33K), (orange, orange, black, red, brown)
  - R30(68K), (blue, grey, black, red, brown)
- Capacitors – C7 (4.7nF) Ceramic; Imprinted: 472

Select All Components First

Before beginning assembly, first select all required resistors, capacitor, and switches. The switches are all identical. But carefully select resistors per their color code and match them to the resistor numbers on the schematic. Resistor values have been selected that allow the microcontroller program to determine which switch or switches have been pressed.

Verify the resistor values with the digital Multimeter if you have a question regarding its color code and value.

C. Procedure

Select components and place onto printed copy of schematic diagram.

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Collision Detection PCB Assembly video.
Insert components into PCB and solder each component in this order:

1) Resistor - R23
2) Capacitor - C7
3) Resistor - R30
4) Switch - K6
5) Resistor - R29
6) Switch - K5
7) Resistor - R28
8) Switch - K4
9) Resistor - R27
10) Switch - K3
11) Resistor - R26
12) Resistor - R32
13) Switch - K2
14) Resistor - R25
15) Switch - K1
16) Resistor - R24

Test the circuit using the Multimeter and record in worksheet.
Compare the test to the prototype test.

D. Assembly Steps:

Installing R23 1 Meg ohm resistor
Power is supplied to the collision detection circuit via R23 from V+ so we will install this resistor first. Bend resistor leads for vertical insertion, insert the resistor into the PCB as indicted matching its printed location. Turn the PCB over and solder the resistor.

Installing C7 4.7 Nanofarad Capacitor]
Locate the position for C7 on the PCB, insert C7. Turn the PCB over and solder C7 leads.

Installing R30 68 K ohm Resistor
Locate the position for R30. Bend the resistor leads for vertical installation, insert R30 into the PCB Turn the PCB over and solder the resistor leads.

Installing K6
Locate the position for K6, note orientation, insert K6, turn the PCB over and solder K6 leads.

Installing R29 33 K ohm Resistor
Locate the position for R29. Bend the resistor leads for vertical installation, insert R29 into the PCB Turn the PCB over and solder the resistor leads.

Installing K5
Locate the position for K5, note orientation, insert K6, turn the PCB over and solder K5 leads.

Installing R28 16 K ohm Resistor
Locate the position for R28. Bend the resistor leads for vertical installation, insert R28 into the PCB Turn the PCB over and solder the resistor leads.

Installing K4
Locate the position for K4, note orientation, insert K4, turn the PCB over and solder K4 leads.

Installing R27 8 K ohm Resistor
Locate the position for R27. Bend the resistor leads for vertical installation, insert R27 into the PCB Turn the PCB over and solder the resistor leads.

Installing K3
Locate the position for K3, note orientation, insert K3, turn the PCB over and solder K3 leads.

Installing R26 2 K ohm Resistor
Locate the position for R26. Bend the resistor leads for vertical installation, insert R26 into the PCB Turn the PCB over and solder the resistor leads.

Installing R32 2 K ohm Resistor

Locate the position for R32. Bend the resistor leads for vertical installation, insert R32 into the PCB Turn the PCB over and solder the resistor leads

Installing K2

Locate the position for K2, note orientation, insert K2, turn the PCB over and solder K2 leads.

Installing R25 2 K ohm Resistor

Locate the position for R25. Bend the resistor leads for vertical installation, insert R25 into the PCB Turn the PCB over and solder the resistor leads

Installing K1

Locate the position for K1, note orientation, insert K1, turn the PCB over and solder K1 leads.

Installing R24 1 K ohm Resistor

Locate the position for R24. Bend the resistor leads for vertical installation, insert R25 into the PCB Turn the PCB over and solder the resistor leads

20. Assemble Motor Control Circuit

The motor control circuit assists the microcontroller in controlling the speed and direction of each motor. This special motor control circuit is also called an H-bridge. The devices that create this circuit control on/off, speed, and direction of rotation of the motors.

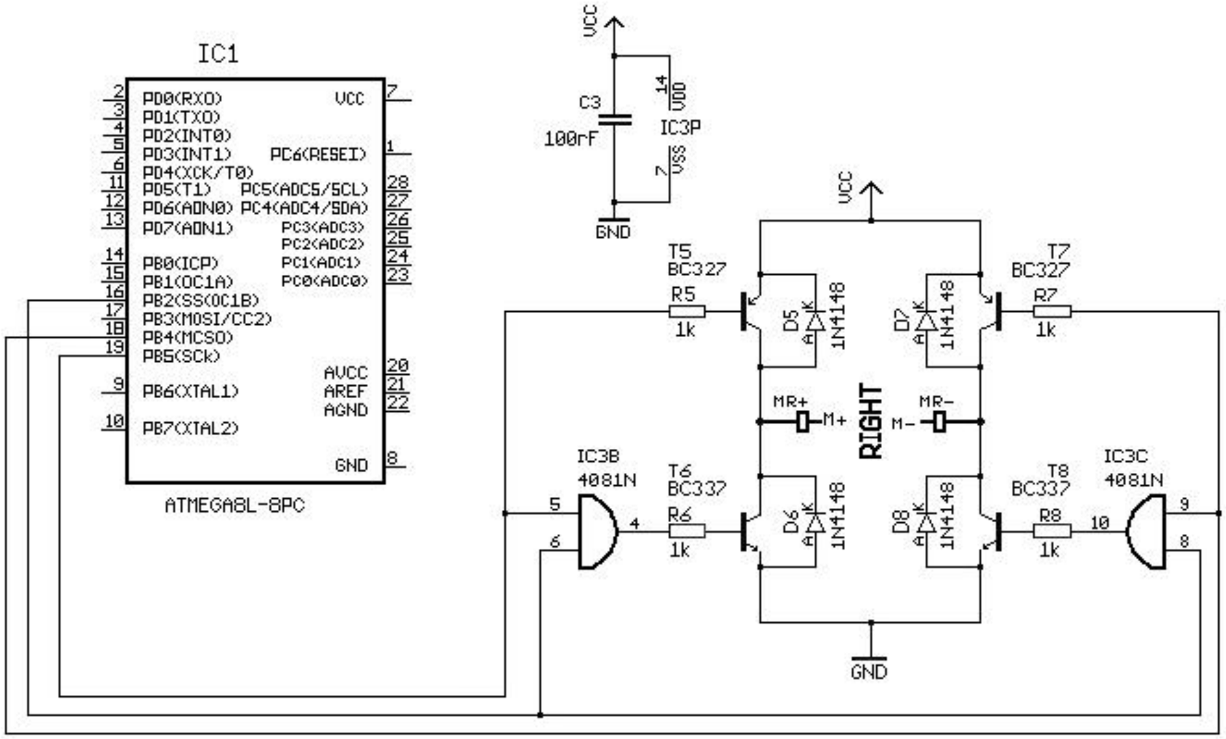
The microcontroller is not able to output a very strong signal, such as required for turning a motor on and off by itself. But if we use a transistor, we can amplify that small signal and control the robot's motors. When we build the circuit and test it, we can see how this works.

The motor control circuit has quite a few components, many of which require close attention to orientation and polarity when building the circuit. Closely observe the part numbers for the transistors as there are two types - PNP and NPN. They must be inserted in their proper positions to work.

Also closely observe the polarity on the diodes associated with each transistor, and double check their orientation before soldering.

A. Schematic Diagram

There are two motor control circuits – a left and a right. Each circuit contains four transistors, four diodes, and four resistors. Although the circuit diagram also shows the four AND gates which are in the Integrated Circuit, we will not be installing the IC in this step.



B. Parts Needed

PCB board from prior assembly
Right Motor Control Parts:

- Transistors :
  - T5, T7: BC327-40 or BC328-40
  - T6, T8: BC337-40 or BC338-40
- Diodes – D5, D6, D7, D8 (Diode 1N4148; Pay attention to polarity!)
- Resistors - R5, R6, R7, R8 : 1KΩ 5% (brown, black, red, gold)
- Integrated Circuits - IC3B, IC3C **(IC3 is NOT Inserted now)**

Left Motor Control Parts:

- Transistors:
  - T1, T3: BC327-40 or BC328-40
  - T2, T4: BC337-40 or BC338-40
- Diodes– D1, D2, D3, D4 (Diode 1N4148; Pay attention to polarity!)
- Resistors- R1, R2, R3, R4: 1KΩ 5% (brown, black, red, gold)
- Integrated Circuits - IC3A, IC3D **(IC3 is NOT Inserted now)**

C. Procedure

Locate all parts and lay them out on a printed schematic diagram of the right and left motor control circuits.

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Motor Control PCB Assembly video. Care in soldering is very important as solder positions may be tight.

Insert components into PCB and solder each component in this order:

Right Motor Control

1) Capacitor - C3
2) Resistor - R5
3) Resistor - R7
4) Resistor - R6
5) Resistor - R8
6) Diode - D6
7) Diode - D8
8) Transistor - T6
9) Transistor - T8
10) Diode - D5
11) Diode - D7
12) Transistor - T5
13) Transistor - T7

Left Motor Control

14) Resistor - R1
15) Resistor - R3
16) Resistor - R2
17) Resistor - R4
18) Diode - D2
19) Diode - D4
20) Transistor - T2
21) Transistor - T4
22) Diode - D1
23) Diode - D3
24) Transistor - T1
25) Transistor - T3

Test the circuit board using the Multimeter and record in worksheet
Compare results to prototype or to anticipated values

D. Assembly Steps

Motor Control Subsystem Assembly and Soldering Estimated Time: 35-45 minutes

Right Motor Control Circuit
First we will assemble the right motor control circuit. We begin with the installation of the capacitor C3

Installing Capacitor C3

Locate the position for Capacitor C3: insert the capacitor leads through the PCB. Solder C3 leads, and trim excess lead lengths.

Installing Four 1K ohm resistors R5, R6, R7, and R8
Next we will install four 1K ohm resistors R5, R6, R7, and R8. Begin by Bending the resistor leads for vertical mounting, then insert each resistor in appropriate locations, Solder resistor leads, trim excess lead lengths.

Installing Four Diodes D5, D6, D7, and D8
All of these diodes are the same. Note that a black band on each diode must be aligned as indicated by a white stripe at each diode location. Insert the four diodes, solder the diode leads, trim excess lead length.

Installing Transistors T6 and T8
We will install transistors in pairs as there are three leads with each transistor and it is easier to track soldering by only installing transistors in pairs. Locate the positions, note the orientation the flat side of the transistor, for T6 and T8, insert the transistors, turn the PCB over; solder the leads, trim the excess lead length.

Installing Transistors T5 and T7
Locate the positions for T5 and T7, , note the orientation the flat side of the transistor, insert the transistors, turn the PCB over; solder the leads, trim the excess lead length.

Left Motor Control Circuit
Installing Four 1K ohm resistors R1, R2, R3, and R4
Next we will install four 1K ohm resistors R1, R2, R3, and R4. Begin by bending the resistor leads for vertical mounting, then insert each resistor in appropriate locations, Solder resistor leads, trim excess lead lengths.

Installing Four Diodes D1, D2, D3, and D4
All of these diodes are the same. Note that a black band on each diode must be aligned as indicated by a white stripe at each diode location. Insert the four diodes, solder the diode leads, trim excess lead length.

Installing Transistors T2 and T4
We will install transistors in pairs as there are three leads with each transistor and it is easier to track soldering by only installing transistors in pairs. Locate the positions, note the orientation the flat side of the transistor, the transistor, for T2 and T4, insert the transistors, turn the PCB over; solder the leads, trim the excess lead length.

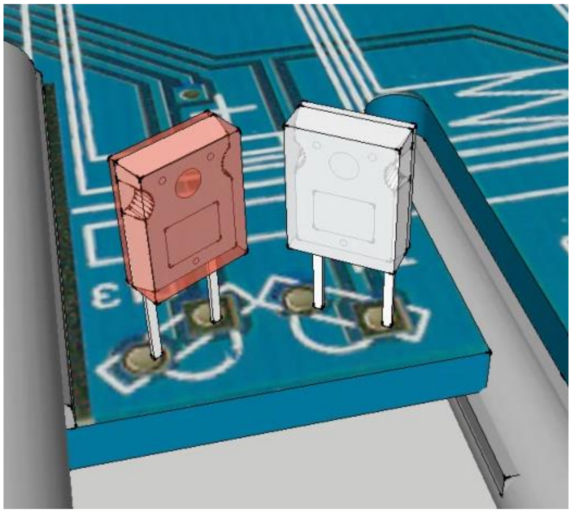
Installing Transistors T1 and T3
Locate the positions for T1 and T3, , note the orientation the flat side of the transistor, insert the transistors, turn the PCB over; solder the leads, trim the excess lead length.

21. Assemble Encoder Circuit

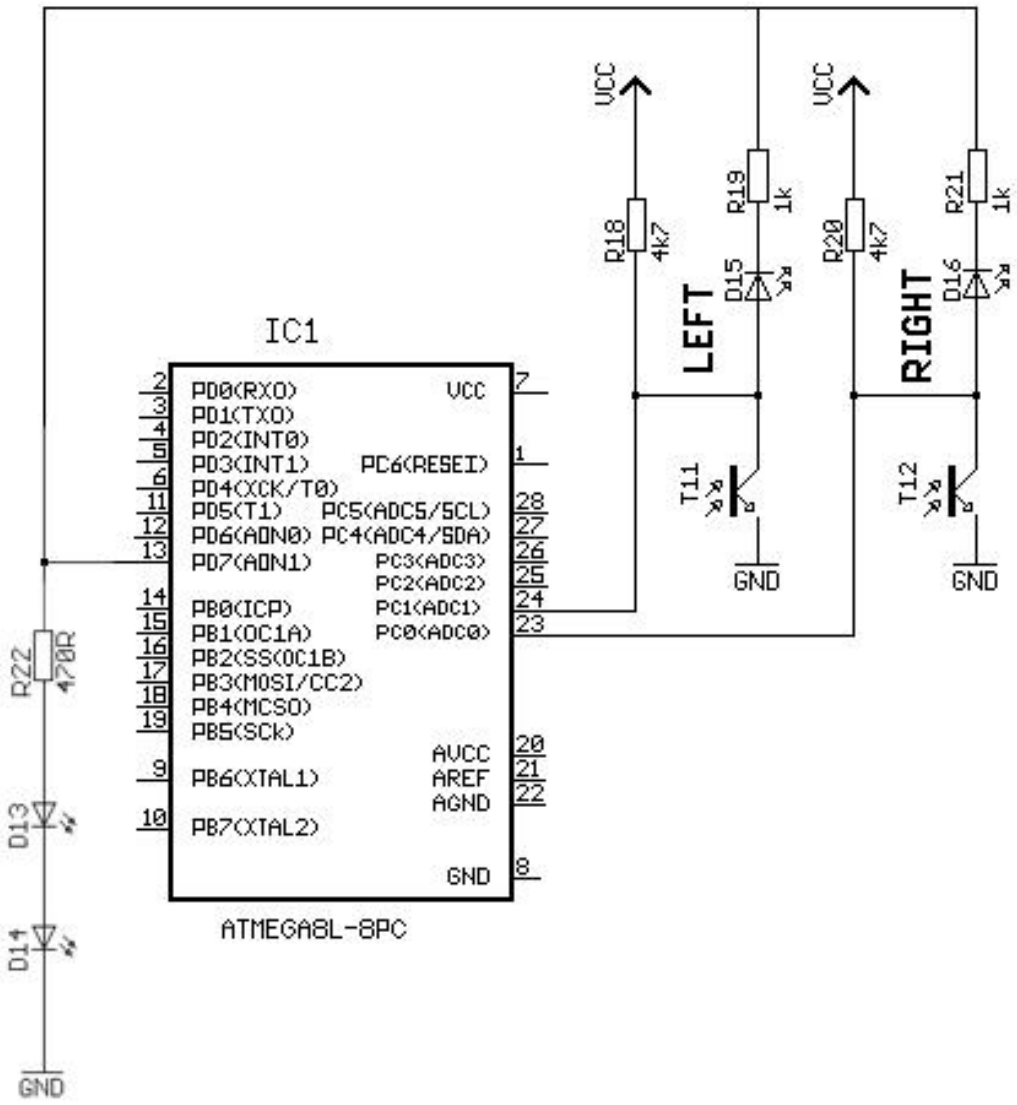
The encoder circuit senses the presence or absence of a black line on a target mounted on one of the ARX gears. The microcontroller uses this information to detect distance and speed of the robot and provide more control. There are two encoder circuits, a right and left, one for each wheel.

The encoder/ odometer circuit depends upon proper orientation and mounting of the infrared LEDs and their respective location and distance to the encoded gear. The encoded gear position may shift depending upon the position of the driven wheel gear. These will not be added until a later. We will discuss this further when assembling the motors and wheels.

Each encoder circuit uses a phototransistor paired with an IR LED. Both the phototransistor and IR LED must be oriented properly. Observe the polarity and double check before soldering. The case of these components should be mounted a short distance from PCB board with legs as shown here (use notch on legs for guidance).



A. Schematic Diagram



B. Parts Needed

- PCB board from prior assembly
- Resistor R18, R20 = 4.7K Ohms (yellow, violet, red, gold)
- Resistor R22 = 470 Ohms (yellow, violet, brown, gold)
- Photo-Transistor T11, T12 = (colorless case) Case must be settled close to PCB; Pay attention to polarity!
- IR LEDs D13, D14 = (pink to rosy case); Case must be settled close to PCB; Pay attention to polarity!

Status LEDs with Resistors (These were already installed with status circuit)

- LEDs = D15, D16 (red) Pay attention to polarity (short leg must be inserted at the mark)
- Resistor R19, R21 = 1K Ohms (brown, black, red, gold)

C. Procedure

Gather parts needed and place on top of a printed copy of the schematic diagram.

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Encoder PCB Assembly video.

Insert components into PCB and solder the components in this order:

1) Resistor - R18
2) Resistor - R22
3) Resistor - R20
4) Transistor - T11
5) Diode - D13
6) Transistor - T12
7) Diode - D14

Test the circuit using the Multimeter
Compare test results to test results from prototype

D. Assembly Steps

First collect all components from inventory. Be sure to confirm all resistor values, when in doubt, check with the Multimeter. The encoder circuit consists of two pairs of phototransistors and diodes, one for each motor and gear assembly.

Installing R18, 4700 ohm Resistor
Bend the resistor leads for vertical mounting, locate R18's position on the PCB, insert the resistor, solder the resistor leads, and trim excess lead length.

Installing R22, 470 ohm resistor
Bend the resistor leads for vertical mounting, locate R22's position on the PCB, insert the resistor, solder the resistor leads, and trim excess lead length.

Installing T11, Phototransistor
Locate T11's position on the PCB, note its orientation the side with a bump is toward the outside of the PCB. Insert T11

Installing D13, LED
Locate D13's position on the PCB, Note its orientation the side with the bump is toward the outside of the PCB. Insert D13.

Solder T11 and D13
Turn the PCB over and solder the leads for T11 and D13, trim the excess lead length.

Installing T12, Phototransistor
Locate T12's position on the PCB, note its orientation the side with a bump is toward the outside of the PCB. Insert T12

Installing D14, LED

Locate D14's position on the PCB, Note its orientation the side with the bump is toward the outside of the PCB. Insert D14.

Solder T12 and D13

Turn the PCB over and solder the leads for T12 and D14, trim the excess lead length.

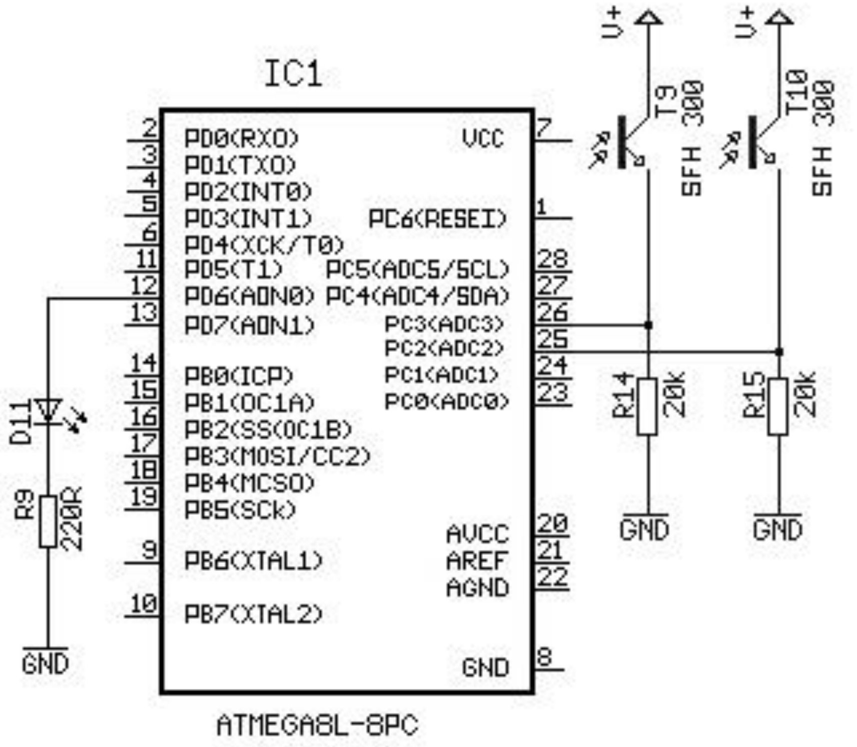
22. Assemble Line Follower Circuit

In a manner similar to the encoder, the line follower circuit uses two phototransistors. These sense light reflected from the line from a single LED. These components are mounted on the front bottom of the circuit board. These solders occur on the top/component side of the board. Care must be taken to not damage any of the other components while soldering.

The LED and phototransistors should not be mounted tightly against the PCB but should stand off slightly - about 1/8 to 1/4 of an inch. Doing so places the LEDs and phototransistors in closer proximity to the line, which makes for easier line following. There are also resistors in this circuit. Once again double check the resistor values with the resistor color chart.

A. Schematic Diagram

The line following circuit consists of resistors R9, 220 ohms and R14, R15 both 20 K ohms, D11 a red LED, and two phototransistors T9 and T10. Each phototransistor is paired with a resistor and inputs to the microcontroller are pins 25 and 26.



B. Parts Needed

- PCB Board from prior assembly
- Resistor- R9 (brown, black, red, gold)
- Resistor- R14, R15 (red, black, orange, gold)

- Diode - D11 LED 5 mm red, red or reddisch case; Pay attention to polarity! (short leg must be inserted at the mark)!
- Transistor- T9, T10 SFH300, Phototransistor 5 mm; Pay attention to polarity! These components have to be placed at some distance from the PCB.

C. Procedure

Gather components and place on top of a printed copy of the schematic diagram.

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Line Follower PCB Assembly video.

Insert components into PCB and solder the components in this order:

1) Resistor- R9 (brown, black, red, gold)
2) Resistor- R14, R15 (red, black, orange, gold)
3) Diode - D11 LED 5 mm red, red or reddisch case; Pay attention to polarity! (short leg must be inserted at the mark)
4) Transistor- T9, T10 SFH300, Phototransistor 5 mm; Pay attention to polarity! These components have to be placed at some distance from the PCB.

Test the circuit using the Multimeter
Compare test results to test results from prototype

D. Assembly Steps

We begin by locating the positions for each resistor, and then bending resistor leads for vertical mounting.

Installing R9, 220 ohms
Locate the location for R9, bend its leads for vertical mounting, insert the resistor, solder its leads, trim excess lead length.

Installing R14, 20 K ohms
Locate the location for R14, bend its leads for vertical mounting, insert the resistor, solder its leads, trim excess lead length.

Installing R15, 20 K ohms
Locate the location for R15, bend its leads for vertical mounting, insert the resistor, solder its leads, trim excess lead length.

Installing D11, T9, and T10
D11, T9, and T10 are the only electronic components inserted from the bottom of the PCB. Locate the position of each of these components by turning the PCB over, note the orientation of each device, the flat side. Insert each component allowing each to stand off from the PCB by about ½ inch, turn the PCB over, solder the leads, trim excess lead length.

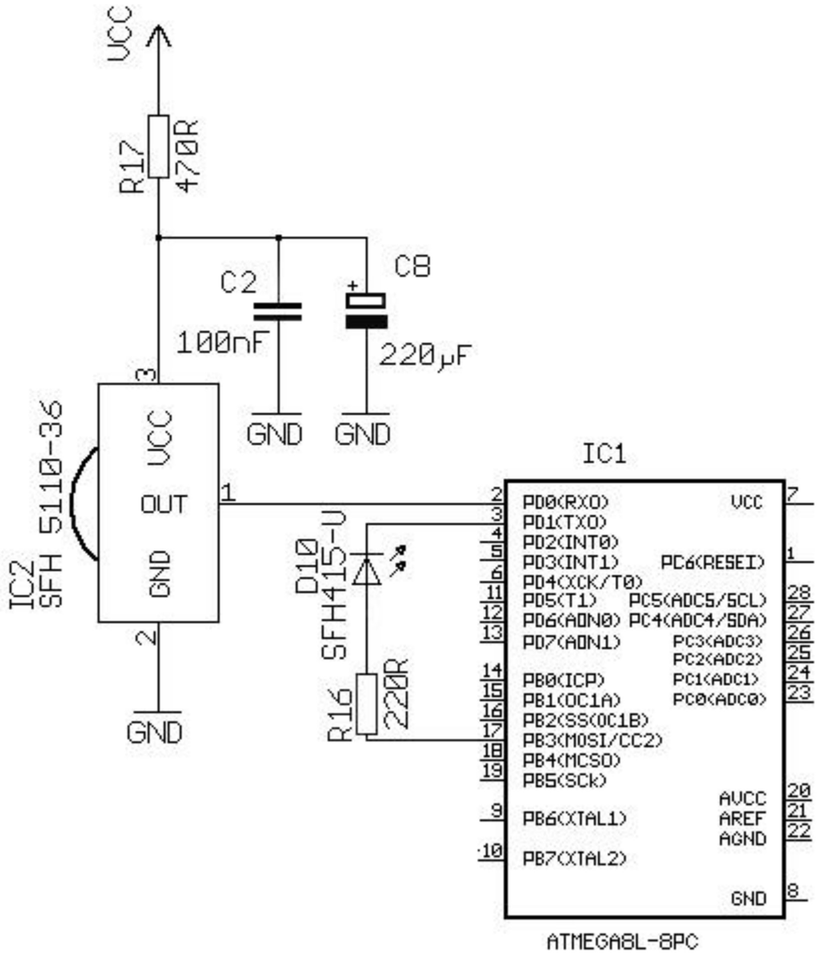
23. Assemble IR Com Link Circuit

The IR Communication Circuit provides a Data link between the robot and a remote PC. This allows programming, remote robot control and data about the robot to be transmitted to the PC. Two primary active components of this circuit are D10, an IR LED which transmits data via Infrared Light, and IC3 an Infrared Integrated Circuit which receives data.

The IR COM circuit requires only a few components, but these are essential for proper robot operation. Take care to assure proper orientation and polarity of all components. There are two capacitors used in the circuit. One is polarity sensitive and the other is not.

Closely observe the orientation of the IR receiver and the IR LED as shown in the video. They are oriented so they are in position to receive the IR signal from the PC. The IR sensor is active only on the side with the bump.

A. Schematic Diagram



B. Parts Needed

- PCB board from prior assembly

- IR LEDS – D10 (SFH 415-U 5mm; black case); Pay attention to polarity! Case must be settled close to PCB.
- Resistors – R16 (220 Ohm), R17 (470 Ohm),
- Integrated Circuit – IC2 (IR Detector) SFH5110-36 Infrared-receiver-IC, bend the legs with long-nose pliers! Pay attention to polarity (the side with dome-shaped curvature must be positioned to the outside)! Caution: electrostatic discharge (ESD) and excessive soldering or heating may damage the part!
- Capacitors – C2 (100 nF), C8 (220µF)

C. Procedure

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the IR COM Link PCB Assembly video.

Insert components into PCB and solder the components in this order:

1) Resistor - R16
2) Resistor - R17
3) Capacitor - C8
4) Capacitor - C2
5) IR LED - D10
6) IC - IC2 (Insert the IC2 Chip last – after all other parts have been soldered)

Caution: IC2: SFH5110-36 - electrostatic discharge (ESD) and excessive soldering or heating may damage the part!

Test the circuit board using the Multimeter
Record measurements and compare to prototype measurements

D. Assembly Steps

Installing R16, 220 ohm Resistor
Bend resistor leads for vertical installation, locate the position for R16, insert the resistor, solder the resistor, and trim excess lead length.

Installing R17, 470 ohm Resistor
Bend resistor leads for vertical installation, locate the position for R17, insert the resistor, solder the resistor, and trim excess lead length.

Installing C8, 220 microfarad capacitor
Observe polarity. Locate the capacitor position. Insert the capacitor, solder the leads, and trim excess lead length.

Installing C2, 100 nano-farad capacitor
Locate the C2's position insert the capacitor, solder the leads, and trim excess lead length.

Installing D10, IR LED
Locate D10's position, observe polarity, insert LED, solder leads, and trim excess lead length.

Installing IC2, IR communication Integrated Circuit
Ground yourself before picking up IC2. Note polarity, bump on IC faces outside of PCB, insert the IC, solder its leads, and trim excess lead length. Use caution - Excess heat from solder can damage the IC.

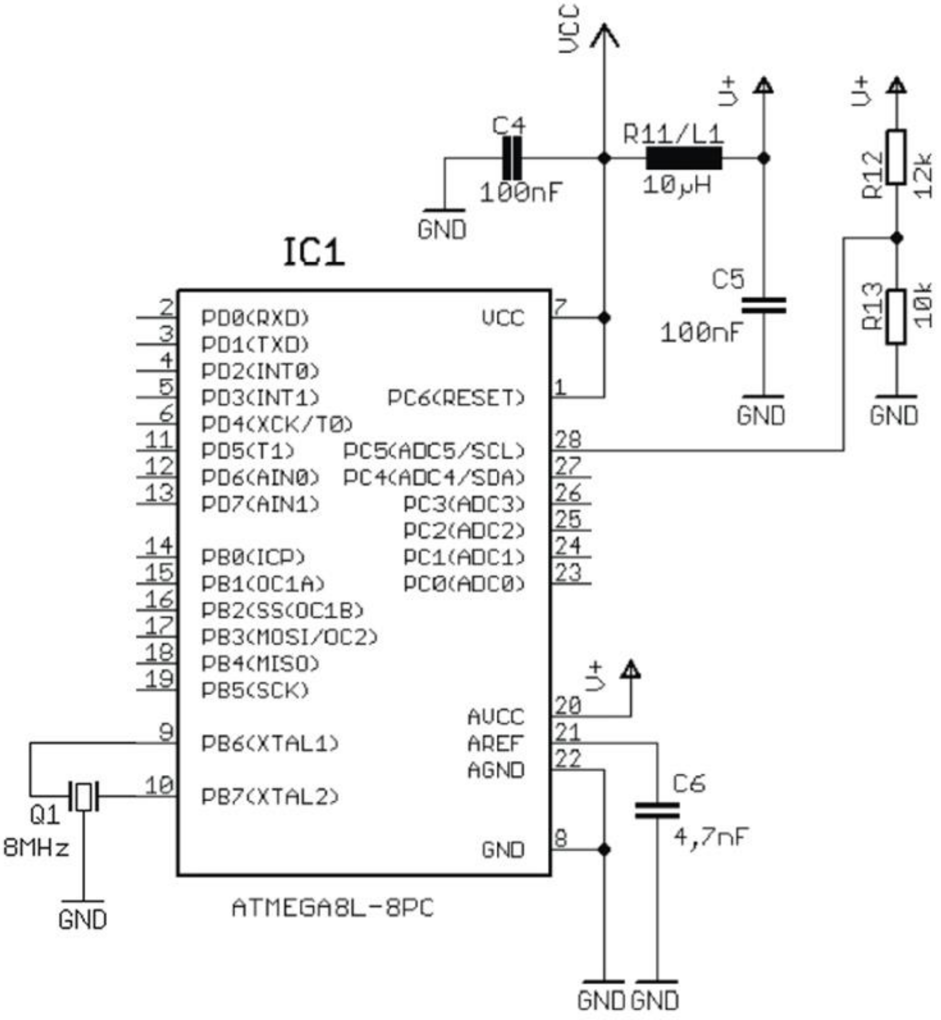
24. Assemble Microcontroller Circuit

For the microcontroller circuit assembly, we install the other components onto the PC board before inserting the Microcontroller into its socket. As discussed in the prototype circuit, take care to ground yourself before touching the microcontroller chip.

Once again it is important to check the resistor values to assure proper circuit operation. This circuit also includes a coil which looks a green resistor. There is a capacitor and other parts which are polarity sensitive. Once again, double check the component orientation prior to soldering. We will also be inserting and soldering the resonator which looks similar to a capacitor, but has three legs instead of two. Observe orientation and polarity.

The battery voltage monitoring is accomplished with a voltage divider using two resistors - a 10K and 12K. Once again verify these resistor values before soldering.

A. Schematic Diagram



B. Parts Needed

- Integrated Circuit – IC1 (ATMEGA8L- 8PC) **(NOT INSTALLED AT THIS POINT)**
- Resonator – Q1: 8 Mhz
- Capacitors:
  - C4, C5: 100nF Ceramic; Imprinted: 104
  - C6: 4,7nF Ceramic; Imprinted: 472
- Coil – R11/L1: a 10µH coil (brown, black, black, silver) which looks like a resistor but is green in color.
- Resistors:
  - R12: 12KΩ 1% (brown, red, black, red, brown)
  - R13: 10KΩ 1% (brown, black, black, red, brown)

C. Procedure

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Microcontroller PCB Assembly video.
Insert components into PCB and solder the components in this order:

1) Resonator - Q1 Resonator 8MHz
2) Capacitor – C6 4,7nF Ceramic; Imprinted: 472
3) Capacitor – C4, C5 100nF Ceramic; Imprinted: 104
4) Coil - R11/L1 an 10µH coil (green color with stripes -brown, black, black, silver).
5) Resistors R12 and R13

DO NOT insert the microcontroller IC chip at this point, we will insert it later.
Caution: IC1: electrostatic discharge (ESD) and excessive soldering or heating may damage the part!
Test the circuit using the Multimeter
Record measurements and compare to prototype measurements

D. Assembly Steps

Finally we are ready to assemble the brains of our robot, that controls all other subsystems, then it is possible to begin looking at the controlling software. We begin by installing the resonator.

Installing Q1, 8 Megahertz Resonator
The coil or resonator is like a system clock, its ticks control the microcontroller's program execution. Locate Q1's position insert Q1, solder its leads, and trim excess lead length.

Installing C6, 4.7 nano-farad Capacitor
Locate C6's position, insert it, solder its leads, and trim excess lead length.

Installing C4, 100 nano-farad Capacitor
Locate C4's position, insert it, solder its leads, and trim excess lead length.

Installing C5, 100 nano-farad Capacitor
Locate C5's position, insert it, solder its leads, and trim excess lead length.

Installing R11/L1, 10 Micro-henry Inductor
Bend L1's leads for vertical installation, insert L1, solder its leads, and trim excess lead length.

Installing R12, (12K) and R13 (10K)
Locate R12's position, insert it, solder its leads, and trim excess lead length.
Locate R13's position, insert it, solder its leads, and trim excess lead length.

25. Assemble Motors and Wheels

Two motor-driven wheels provide propulsion, or movement for the ARX robot. One wheel is mounted on either side of the ARX chassis. The PCB board acts as the body of the robot and holds all the mechanical parts in place. When both wheels are turning to move forward at the same speed, then the ARX moves in a straight line. Turning the ARX is accomplished by moving one wheel at a different speed than the other.

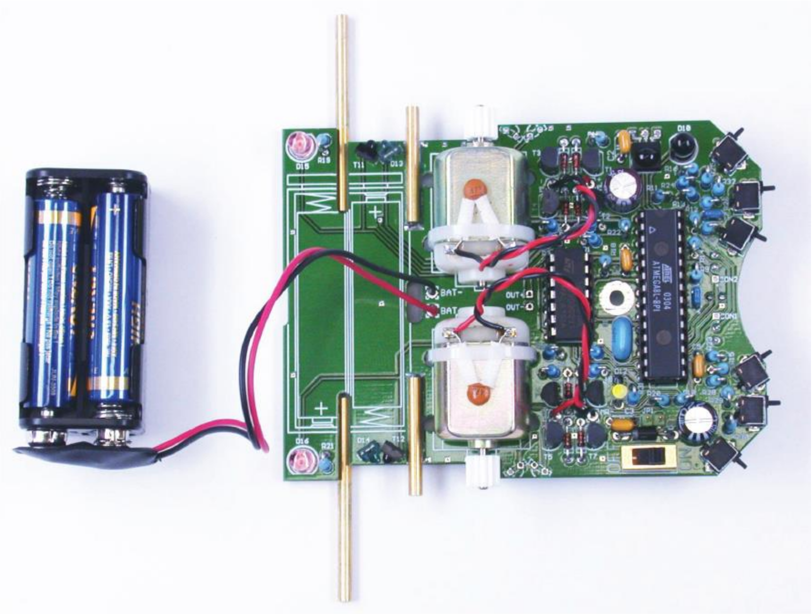
This step involves the primary mechanical components for operation of the robot. We will install the motors, gears, and wheels. We will also wire the two drive motors. We will also be mounting the encoder - which is a sticker applied to one gear. The encoder gear is the first gear driven by the motor. It is inserted on the short axle and depends upon the orientation of the wheel gear for its position.

The long axle holds the final driven gear and the wheel retention hub. When properly installed, the wheels will move freely without binding.

Finally we add the motors. Two short wires are used to attach each motor to the PCB board - one red, one black. Observe polarity and take care in stripping and soldering the wires. The motors are attached to the PCB board with plastic ties. It is important that they match with the gears and be tightly secured. Observe the orientation and positioning of the gears and motors.

A. Motors

The ARX is propelled using two DC motors attached to a gear box connected to wheels attached to the PCB with axles. These parts need some prior assembly and then are placed onto the PCB.



1) Motorpinion

Motor power is transmitted to the gears by small cogwheels which MAY have to be pressed onto the motor axles. ***NOTE: FOR MOST KITS THIS CAN BE SKIPPED, The motors already have the cogwheel in place!***

Place the rounded side of the cogwheel on a soft flat surface (such as a cardboard on a table) and press the motor axle gently into the drilling hole of the cogwheel without using too much force. A small hammer or a bench vice may be needed to press the axle of the motor fully onto the cogwheel. Take care not to press the axle and apply any force to the case or the bearings during this procedure.

2) Motors

The motors are attached to the top of the PCB with plastic ties. The small cogwheels are to the outside of the PCB, and placed so they will align with the power transmission gears. The plastic ties are not attached and tightened until the gears are aligned.

ARX's motors are designed to operate at around 3 to 7 VDC, with DC batteries. They are brushed DC motors. The simple electric motors contain a turnable armature, a fixed permanent magnet, and a commutator. The armature uses three coils and is located between the permanent magnet's poles.

The specifications of the motor are:

- Voltage range: 3.0 to 12.0 V
- Speed range: 12,400 to17,500 RpM
- Current range: 0.12 to 0.34 Amps
- Torque: 0.10 Ncm
- Power output: 1.26 W
- Efficiency: 30%
- Weight: 18 g

B. Power Transmission

The robot drive train or transmission consists of the two driven wheels which are connected via gears to two drive motors. There are two sets of gears which combine to reduce motor speed and provide good tractive torque at the driven wheels.

The gear train consists of two sets of gears arranged in a special configuration on two axles. A small pinion gear on the motor shaft is engaging a larger gear.

1) Ball Support

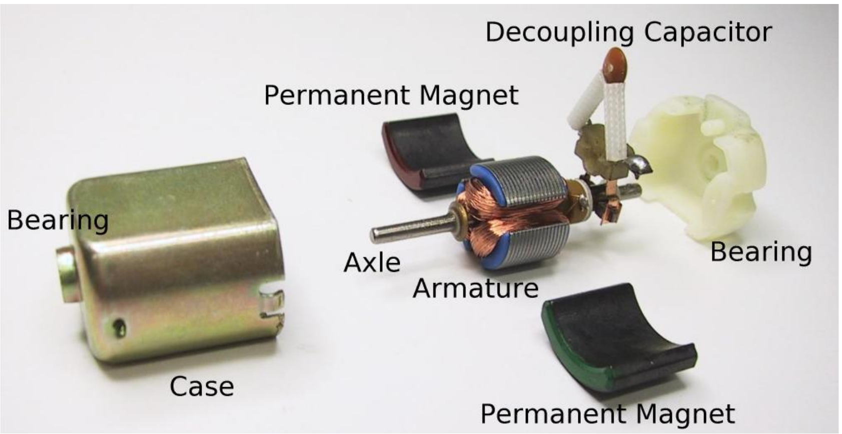
The ARX ASURO has been designed to slide on half of a ping pong ball, which has to be prepared before installation.

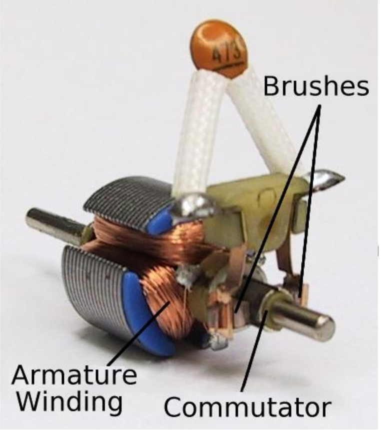
A sliding semi-sphere (½ of a ping pong ball), is mounted under the PCB and slides easily on most surfaces thus allowing the driving wheels to pull the chassis around.

NOTE: We will NOT Attach the ball support during this step, it will be done later after we have tested the operation of the PCB board.

2) Motor Operation

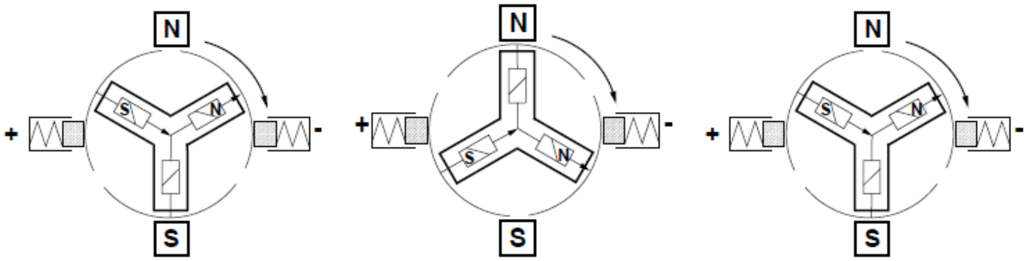
How does the motor work? The ARX ASURO's simple electric motor contains a turnable armature on the axle, a fixed permanent magnet and a commutator. The armature uses three coils and is located between the permanent magnet's poles.





The illustration below explains the rotating principle. Thin lines mark the electric, thick lines mark the magnetic-mechanical elements. The inner side displays the armature's position and the surrounding elements display the commutator, consisting of a slip ring and the brushes. The sketch also displays the permanent magnet's North Pole and South Pole. The coil's windings have been chosen to guide the current from the outside inward, generating a South Pole at the outside of the armature.

The armature's North Pole will be repelled by the permanent magnet's North Pole and be attracted by its south pole. Equally the armature's South Pole will be repelled by the permanent magnet's South Pole and be attracted by its north pole. The motor will now start turning to the right, until the commutator shortly before reaching a stable position reverses the coil's current flow, forcing the armature to continue turning to the right.



3) Torque

What is Torque and how does a motor generate torque? *A moving anchor in the permanent magnetic's field induces a voltage in the anchor's coils, which resists the applied voltage. The opposing voltage will increase with motor speed until it reaches the motor's applied voltage. Theoretically -*

in an ideal motor system- this principle causes the motor speed to increase linearly with motor voltage. Increasing the coil's currents will equally increase the magnetic field and the motor power. For this reason an ideal motor's rotational power (called "torque") will be proportional to the motor current.

***Torque**, **moment** or **moment of force** is the tendency of a force to rotate an object about an axis, fulcrum, or pivot. Just as a force is a push or a pull, a torque can be thought of as a twist to an object. (source: http://en.wikipedia.org/wiki/Torque)*

4) Electromagnetism

***What is Electromagnetism?** Electromagnetism, or the **electromagnetic force** is one of the four fundamental interactions in nature, the other three being the strong interaction, the weak interaction, and gravitation. This force is described by electromagnetic fields, and has innumerable physical instances including the interaction of electrically charged particles and the interaction of uncharged magnetic force fields with electrical conductors. (source: http://en.wikipedia.org/wiki/Electromagnetism)*

5) Gearbox

Transformation of rotational speed or torque requires gears. To move the ARX ASURO robot, the motor's torque requires a dual stage gear. Critical gear parameters are the gear transformation and the gear wheel shape or module. The gear module is the ratio of the wheel pitch circle diameter to the number of teeth. Increasing the gear module number results in bigger teeth. Of course the teeth sizes and gear modules in concurrent gear wheels must be identical.

6) Gear Ratios and Formulas

The gear train consists of two identical gears arranged in a dual-stage configuration on two axles. This provides transformation of rotational speed or torque from the motors to the wheels.

There are mathematical formulas and gear parameters that are used to determine the effectiveness of the power transmission which include:

- The gear module *m* is the ratio of the wheel diameter to the number of teeth. Increasing *m* results in bigger teeth. The teeth sizes and gear modules in concurrent gear wheels must be identical.
- The distance *d* between axles.
- The number of teeth *z* on the gear.
- The distance *d* between axles may be derived from the number of teeth *z* with this formula:
  $d = ((z \text{ propelling} + z \text{ propelled}) / 2) * m[mm]$

- The gear ratio *i* is the quotient of the teeth numbers *z:*
  $i= z \text{ propelled} / z \text{ propelling}$
- A higher gear will decrease the speed *n* by:
  $n \text{ propelled} = n \text{ propelling} / i$
- Combined gear ratios are obtained by multiplication:
  $itotal = i1 * i2 * i3....$

A pinion gear on the motor shaft has 10 teeth (propelling gear) and is engaging a gear that has 50 teeth (propelled gear), thus giving it, using the formula *I = z propelled / z propelling, we get 50/10* which reduces to a gear reduction ratio of 5:1

On the back side of the large 50 teeth gear, is attached the first driven gear that has 12 teeth. This smaller gear (propelling gear) engages a second larger gear that has 50 teeth (propelled gear) giving a ratio of 50/12 or approximately a 4:1 reduction. This second larger gear is attached to the wheel.

Thus using the formula *itotal = i1 * i2, (itotal = 5:1 * 4:1),* our total reduction from the motor to the driven wheel is about 20:1. This means that for every 20 rotations of the motor shaft we get 1 rotation of the driven wheel.

C. Parts Needed

3D CAD drawings have been provided which illustrate all the mechanical parts of the ARX. These drawings have step by step instructions for assembly and can be zoomed, panned, and rotated for a clear view of how parts are attached.

- 2x Motors Type Igarashi 2025-02
- 4x Cable binder
- 2x Encoder sticker
- 2x Cogwheel 10/50 cogs; 3.1mm drilling hole Module 0.5
- 2x Cogwheel 12/50 cogs; 3.1mm drilling hole Module 0.5
- 2x Pinion gear 10 (oder 12) cogs; drilling hole 1.9mm Module 0.5
- 2x Rubber tires 38mm

D. Procedure

Assemble and wire the circuit using the PCB board and parts following the schematic diagram and steps illustrated in the Motor PCB Assembly video.

Assemble the parts in this sequence:

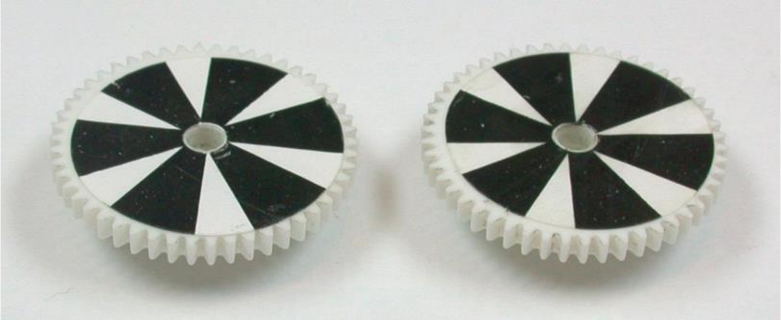
1. Attach wheel sensors for encoder
2. Attach wires to motors
3. Attach motors to PCB Board
4. Assemble right gears and wheel
5. Assemble left gears and wheel
6. Test operation of motors, gears and wheels

E. Assembly Steps

1) Attach Wheel Sensors for Encoder

The 50 tooth/10 tooth gear set is used as an encoder. There are adhesive-backed cogwheel markers with 6 black and white segments used with the ARX ASURO. Adhere the odometer markers onto each of the larger 50 tooth gears.

For each rotation of the gear, the black spots on the label will generate six pulses from the phototransistor which monitors transitions from light to dark of the encoder label. The count of the pulses is used to determine the distance traveled.



2) Attach Wires to Motors

Attach black wire to metal pin on negative side of motor and solder. Attach red wire to metal pin on positive side of motor (marked with red) and solder.

3) Attach Motors

The ARX ASURO engine consists of two electric motors, each driving a wheel by means of a dual stage gearbox. Place the motors on top of the PCB in the spaces indicated. Place plastic ties around motors and under PCB. Do not tighten securely until gears and wheels have been assembled and oriented with the motors.

4) Assemble Gears

Two sets of gears are used to drive the ARX ASURO wheels. On the back side of the large gear, is attached the first driven gear which is smaller. This smaller gear engages a second larger gear. This second larger gear has a smaller gear which is where the wheel is attached. This assembly process can be a bit tricky.

1. Place a large gear (with the encoder on the side closest to the PCB) onto the shorter axle. Align this wheel to the cog gear on the motor shaft.
2. Next install the second large gear (with the small gear attached) on the longer shaft. Once again the larger gear is closest to the PCB board. Align the teeth of the larger gear with the teeth on the small gear previously installed on the shorter axle.
3. After all the gears are aligned, then tighten down the plastic ties on the motor to hold the gear assembly with the motors in place. Repeat this process for the other side.

5) Attach Wheels

Grease the axles slightly. Place the cogwheels with its fine black and white markings onto the short axles. The tires combined with the cogwheels are now placed onto the long axles and secured by a locking ring.

Insert the white plastic part into the black wheel and press down.
The wheels are pressed onto to the small gear on the larger axle.
After attaching, test turning of the wheels. The cogwheels must fit exactly and the wheels rotate easily. If the wheels don't rotate easily, either the gear assembly or the axles were not installed properly. First try adjusting the gear assembly, but if that does not work, then the axles will need to be de-soldered and re-attached.

Once the wheel and gear assembly has been aligned, the small brass retaining hub is inserted onto the axle, pushing against the wheel. Finally, the retaining hub set screw is tightened down to hold the wheel assembly in place.

6) Mount the Motors

If the position of the motor has been checked, keep the system tightly fixed while you put a small amount of instant glue between motor and PCB.
After fixture with instant glue the cable ties are applied at the outside of the PCB.

26. Install ICs and Test Operation

We are now at the final stages of assembly. We will insert the IC chips and test the operation of the robot. If we have followed the steps for each subassembly, this process should go smoothly.

The final step in the assembly process is the insertion of the microcontroller and the quad AND gate ICs into their sockets. Care must be taken to assure that all leads are inserted into their proper holes and that none are bent or left unattached. As discussed in the prototype, we ground ourselves before touching the ICs to avoid static charge.

Once the ICs are installed, we are ready to test the microcontroller and the circuits on the PC Board. We insert batteries, and attach the battery pack to the PCB with the removable black plastic tie.

Depending upon whether using rechargeable or non-rechargeable batteries, either remove or leave in place the power supply jumper. Next we turn on the power supply switch on the robot. The microcontroller will automatically begin running a test sequence. We will observe each step of the test sequence and document what happens. This test sequence will allow us to verify the operation of most of the components on the PCB.

We also set up the IR communication on the PC and observe the test sequence. Each test name is displayed on the PC terminal prior to starting that part of test. We observe and document the results of the test, following the instructions in the activity.

A. Procedure

There are several steps here, and a specific test sequence has been determined to assure successful operation of all parts. Overall, this is what we will do:

1. Install the hyperterminal software and setup the IR device on the PC
2. Install IC1 chip (the microcontroller)
3. Install IC3 chip (the AND gate)
4. Place ARX on something so the wheels don't touch the ground (such as a battery case)
5. Open the hyperterminal program and establish a connection between the PC and the robot
6. Turn on power to the robot and it will run the pre-installed test program
7. Follow along with the test sequence. If errors occur, turn off the robot to stop the test program and use the error checking section to debug.
8. Repeat steps 5 and 6 until all errors have been resolved.

B. Parts Needed

- Hyperterminal software
- IR dongle for PC (associated Com software should automatically download)
- PCB board from prior assembly
- 1x Processor ATmega 8L-8PC (preprogrammed) - IC1
- 1x Integrated circuit CD 4081BE – IC3 (AND Gate)

- Power supply and charged batteries

C. USB IR Transceiver Setup

- Install USB and Hyperterminal software following directions in the separate software installation instructions
- Launch Windows Hyperterminal software

D. Install Integrated Circuits (ICs)

1. See caution below before picking up the microcontroller IC. Orient the IC using the notch at the end – aligning the notch to the notch in the socket. (See video)
2. If the IC does not insert easily, sometimes the legs of an IC have to be bent carefully in order to insert all legs into the contact holes of the socket. This can be done easily if you take the IC and press the side with all legs on the row parallel on the surface of a flat table.
3. After installing the microcontroller IC, insert the smaller IC (*IC3 CD408)*
4. Confirm USB IR transceiver is detected

Caution: *Processor IC1 ATmega8 and Gate IC3 CD4081both are electrostatically sensitive devices. They might be destroyed by touching them, if you do have charged yourself electrostatically, which may easily occur by walking over a non-conductive floor.*

Before working with these parts, it is a good idea to wear a grounding bracelet or at least to touch a metallic case of grounded equipment or a heating radiator.

E. Run Test Sequence

Depending upon whether using rechargeable or non-rechargeable batteries, either remove or leave in place the power supply jumper. Next we turn on the power supply switch on the robot. When the power is turned on the robot, the pre-loaded test program will begin. This is the sequence of the test program:

1. Testing LEDs

The Status-LED (D12) shortly lights orange and the Back LEDs (D15, D16) also glow, but these glow only dimly. If one of these signs does not show, switch OFF immediately and start error checking and removal (see Error Checking below).

You just have seen the boot-sequence of the ARX ASURO. In the next phase all display-elements will be checked subsequently at an interval of 3 seconds and in the following order:

- Status-LED (D12) green
- Status-LED (D12) red
- Front-LED (D11) on the bottom side of the ARX ASURO
- Back-LED (D15) left
- Back-LED (D16) right
- All LEDs light together

If an error occurs, switch OFF ARX ASURO immediately and start error checking and removal as all display elements play an important role for all following tests.

2. Testing Phototransistors

After finishing the display tests the Status-LED (D12) should light in green, indicating clearly the Phototransistors at the bottom side of ARX ASURO and responsible for line tracing, are now being tested for about 10 seconds.

While lighting the Phototransistors (T9, T10), the accompanying Back-LEDs (D15, D16) will be turned on and dim as soon as the light is switched off (right phototransistor(T10) → right back-LED (D16); left phototransistor(T9) → left back-LED (D15)). Everything is OK, if the back-LEDs do not go out completely after removing the light and continue glowing dimly. If an error occurs, the self test may be continued anyway. Error removal may be postponed.

3. Testing Switches

ARX ASURO is immobile and all displays are dark. That is a good omen! Now we are ready to check the switches for about 15 seconds. Just press each of the switches, to see if something happens.

The result should be:
K1 → Status-LED (D12) is switched on green
K2 → Status-LED (D12) is switched on red
K3 → Front-LED at the bottom side is switched on
K4 → Back-LED left (D15)
K5 → Back-LED right (D16)
K6 → Motor left starts running (If the motor does not start running, the self test may be continued. The motors will be checked separately later and an error in this segment may be removed in the tests described later).

Pressing several switches will initiate an equivalent combination of signals. If an error occurs, the self test may be continued anyway. Error removal may be postponed.

4. Testing Odometer

Now the line tracer-LED (D11) starts lighting. This signal indicates the beginning of the next test (15 seconds), in which both odometers (each consisting of a LED and a phototransistor) are being checked. Holding a white sheet of paper in front of the sensors will result in lighting the status- LED. First of all hold a sheet of paper in front of the left sensor (T11). → Status-LED (D12) lights green. Then hold a sheet of paper in front of the right sensor (T12) → Status-LED (D12) lights red.

Removing the sheet will result in switching off the status-LED. If the on/off-thresholds are active on the left and on the right side, the odometers are OK. If an error occurs, the self test may be continued anyway. Error removal may be postponed.

5. Testing Motors

Both back-LEDs (D15, D16) are switched on, indicating the last test, which lasts ca. 15 seconds.
The motors will be checked intensively. The left motor is started in forward direction from zero speed to maximum speed and back to zero again. Then the direction will be reversed and speed is accelerated from zero to maximum and back again to zero. The motor on the right side will be checked the same way. After checking the single motors both will be checked simultaneously.

Again we will repeat the remarks only once more: If an error occurs, the self test may be continued anyway. Error removal may be postponed.

6. Testing IR Interface

If the status-LED lights intermittently yellow, the last test has been started (ca. 15 seconds), in which the IR interface transmits and receives data. In order to receive these data, the previously assembled IR-Tranceiver has to be connected to the PC and a terminal program, eg. the Windows terminal program "Hyperterminal" may be used for this purpose. The configuration will be the same as at the testprocedure for the IR-Tranceiver.

At the receipt of symbols, ARX ASURO will answer with a symbol next following in the alphabet. If within a fixed time limit no symbols have been received, ARX ASURO transmits a 'T'. Any sent symbol will switch on the RED status-LED parallel to the green status-LED, which results in an orange flashing color.

If an optical connection between ARX ASURO and the Infrared-Transceiver has been arranged (which requires a maximal distance of ca. 50 cm), the terminal program will show up a symbol 'T' regularly or alternatively the button you may have activated at the PC's keyboard (as it has been transmitted by the transceiver), followed by the symbol next in the alphabet, eg.:

Key "e" is activated → terminal program reports "ffffff"
Key "j" is activated → terminal program reports "kkkkk"
Key "3" is activated → terminal program reports "4444"

F. Error Checking

The following are some general error checking procedures to follow when errors occur during the test sequence.

1) A display element does not work

Has the processor been inserted correctly? (Check Polarity!)

Status-LED D12 does not work
Check polarisation of LED D12.
Check resistors R10, R31
470Ω (yellow, violet, brown, gold)

A simple check may be done the following way: remove the processor (IC1) and connect by wire Pin7 (VCC) and Pin14 (Status-LED will be activated green) respectively Pin4 (Status-LED will be activated red). If this test is successful, (1) either the processor may be defective, or (2) a track of the PCB may be broken / interrupted.

Front-LED D11 does not work
Check polarisation of LED D11.
Check resistor R9
220Ω (red, red, brown, gold)
A simple check may be done the following way: remove the processor (IC1) and connect by wire Pin7 (VCC) and Pin12 (Front-LED will be activated red). If this test is successful, (1) either the processor may be defective, or (2) a track of the PCB may be broken / interrupted.

Left back-LED D15 does not work
Check polarization of LED D15.
Check resistors R19, R18.
1KΩ (brown, black, red, gold)
4,7KΩ (yellow, violet, red, gold)
A simple check may be done the following way: remove the processor (IC1) and connect by wire Pin7 (VCC) and Pin24 (left back-LED will be activated red). If this test is successful, (1) either the processor may be defective, or (2) a track of the PCB may be broken / interrupted.

Right back-LED D16 does not work
Check polarization of LED D16.
Check resistors R21, R20.
1kΩ (brown, black, red, gold)
4,7kΩ (yellow, violet, red, gold)
A simple check may be done the following way: remove the processor (IC1) and connect by wire Pin7 (VCC) and Pin23 (right back-LED will be activated red). If this test is successful, (1) either the processor may be defective, or (2) a track of the PCB may be broken / interrupted.

2) Line tracer sensor (T9, T10) does not work

Check polarization of T9, T10.
Check resistors R14, R15.
20KΩ (red, black, orange, gold)
Check, if R15, R23 and R28 have not been inserted in a wrong position!
Check the sensor signal at Pin25 respectively Pin26 with a multimeter after removing the processor (dark ≈ 0V, light ≈ VCC).

3) A switch does not work

A combination of switches has been activated
Check R12 and R13
12KΩ, (brown, red, black, red, brown)
10KΩ, (brown, black, black, red, brown)
Check R24, R25, R26, R27, R28, R29, R30, R32 as well!
1KΩ (brown, black, black, brown, brown)
2KΩ (red, black, black, brown, brown)
8,2KΩ (grey, red, black, brown, brown)
16KΩ (brown, blue, black, red , brown)
33KΩ (orange, orange, black, red, brown)
68KΩ (blue, grey, black, red, brown)
2KΩ (red, black, black, brown, brown)

The display reacts as if switches have been interchanged
Resistors to the switches have been interchanged
Check R24, R25, R26, R27, R28, R29, R30, R32 as well!
1KΩ (brown, black, black, brown, brown)
2KΩ (red, black, black, brown, brown)
8,2KΩ (grey, red, black, brown, brown)
16KΩ (brown, blue, black, red , brown)
33KΩ (orange, orange, black, red, brown)
68KΩ (blue, grey, black, red, brown)

Things still do not work well
Check R23, R24. Check R12, R13 and check C7 as well!
1MΩ (brown , black, green, gold)
1KΩ (brown, black, black, brown, brown)
12KΩ, (brown, red, black, red, brown)
10KΩ, (brown, black, black, red, brown)
220µF/10V

27. Attach Ball and See Robot Run

A sliding semi-sphere (½ of a ping pong ball), is mounted under the PCB and slides easily on most surfaces thus allowing the driving wheels to pull the chassis around.

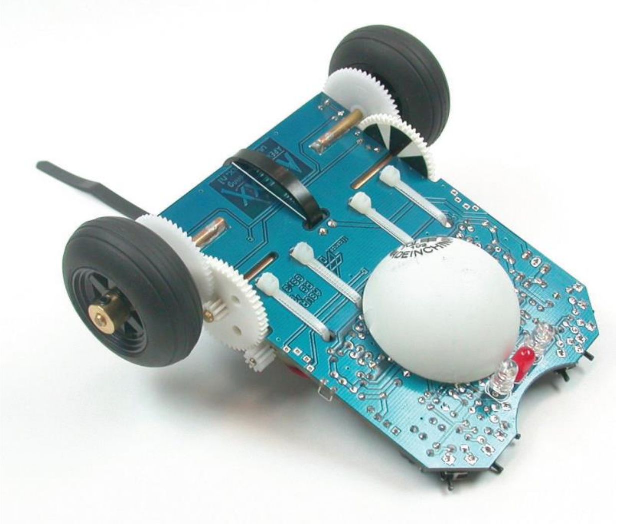
The final step in robot assembly is to attach one half of a table tennis ball to the bottom of the PCB board. This hemisphere provides a third point of support when combined with the two driven wheels. It also slides easily on most surfaces and allows smooth robot movement.

We will be using a hand saw to cut the ball in half and glue to attach it to the PCB board. It is important to have finalized all debugging of the PCB board before gluing the ball on.

After the ball is attached, we are ready to move on to the software installation and testing the operation of the robot.

NOTE: It is VERY IMPORTANT that all debugging of the board has taken place before proceeding with this step. The ball will cover up many of the solder locations for components. It will have to be removed (which is not easy to do) to de-solder or solder components.

A. Pictorial Representation



B. Parts Needed

- PCB board from prior assembly
- Ping Pong Ball
- Tool for cutting ball – hand saw or sharp knife
- Sandpaper
- Glue

C. Procedure

The ARX ASURO has been designed to slide on half of a ping pong ball, which has to be prepared before installation.

Cut the Ball

Take the ball which came with the ARX ASURO kit. If it does not have a line, draw a line to mark the half-way point. Tape it down to the table using clear plastic tape - putting the tape over the line. Then cut thru the tape into two halves with a saw or a blade. You will probably need to turn the ball at least once and tape it down again. (see video)

Note: Do not use an electric saw or electrical knife because the ball may be set afire very easily.

Clean up the cut edges on the ball with sandpaper.

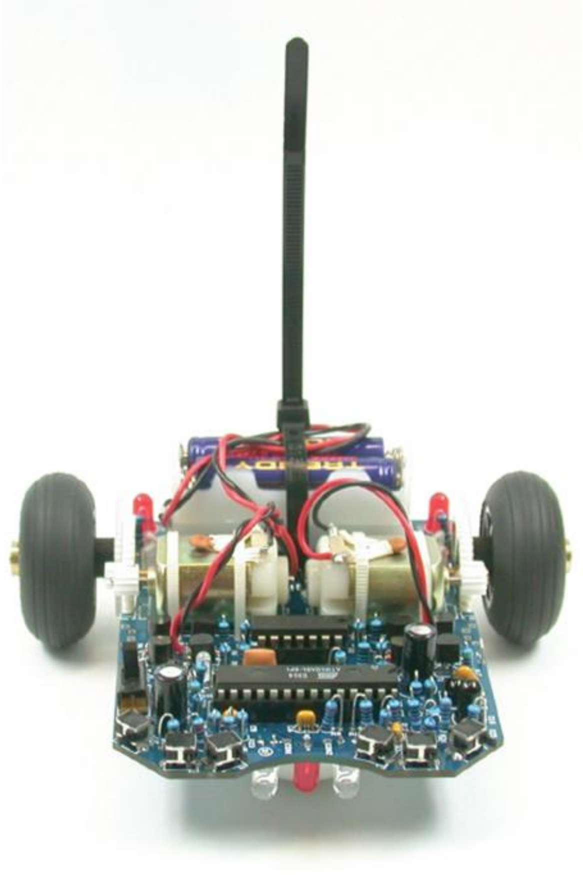
Attach Hemisphere to Robot

Use glue to attach one half of the ball to the bottom side of the PCB so one edge is just behind the line following LEDs. (see picture above)

Congratulations! Your ARX ASURO Robot is now completely assembled!

Section 4: Programming

- *Introduction to C Programming*
- *C Programming Essentials*
- *ARX ASURO Functions*
- *Our First Program*
- *IR Communication*
- *Controlling LEDs*
- *Driving – Controlling the motors*
- *Line Following*
- *Object Detection and Avoidance*
- *More Control with Encoders*
- *Using the A/D Converter*



28. Introduction to C Programming

Having completed the assembly process, we are now ready to begin programming the ARX ASURO robot. In Section 4 we will use the software tools on a PC to create C programs and transfer the programs to the ARX ASURO robot. We will also use the hyper-terminal program to communicate between the PC and the robot.

We will start with easy tasks such as communicating with the robot via IR and flashing LEDs and then advance to moving the robot, following lines, avoiding obstacles, and then move to advanced topics of using the A/D Converter for obtaining information from the encoder and reading battery voltage.

A. Activity

The first programming activity involves installing the software and using a test program to discover the programming process. The video for this activity also includes an overview of all the programs that we will be creating and the special functions we will use to control the robot. Follow the detailed instructions in the activity.

B. Install the Software

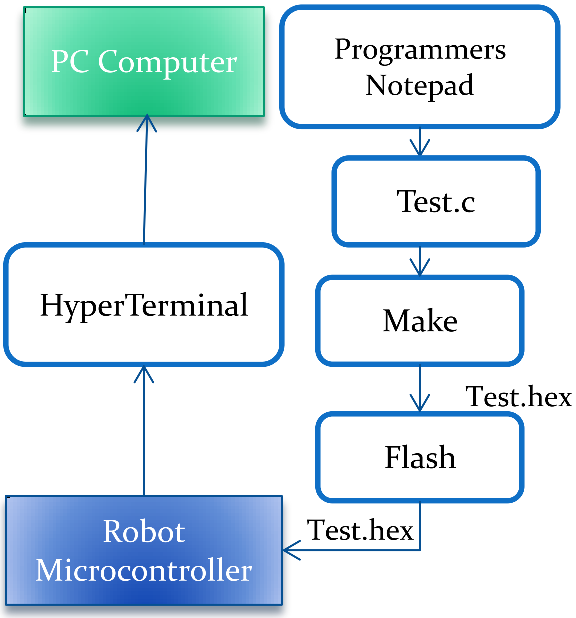
Follow the directions in the separate software installation book to install the software.
The installation procedure will install the following:

- A program editor (Programmers Notepad 2, PN2) and a Compiler (WinAVR).
- Flash-Tool, a program to transmit your own program to ARX ASURO.
- Example program(s) (copied to your hard disk).
- In program editor (PN2), a menu input for Make and Clean batch files is made.

C. Run the Test Program

Follow the instructions in the activity to open, make, flash and run the test program. This is what we will do:

- Open a test program – 'test.c'
- Use Make to compile test program and create 'test.hex'.
- Use Flash to transfer the 'test.hex' file to the robot.
- Turn on the robot to run the test program.



D. Getting Started with C Language

We are not teaching C programming, but are providing enough instruction so that if you are new to C you can still complete the activities. The next section – C Programming Essentials - can be used as a reference to look up commands and syntax for unfamiliar items in the sample programs.

The C Reference Section in the Appendix of this book includes additional C language tips as well as a few advanced topics that are not required for completing the activities. They are included for those students who are advanced C programmers and want to learn more about direct interface with the ports and the A/D Converter.

When you visit a country for a short period of time it is not necessary to be fluent in the language. It is however, convenient to be able to ask directions, order a meal, ask for a bathroom, etc. Likewise, for the programming activities it is not required that you fully understand C language, although it is beneficial. We will cover the essential elements required to allow you to control the robot. We have included additional C reference information in the Appendix of this book.

If you want to learn more, you will easily find a great deal of educational material about C programming on the Internet and in books. The essential elements we share will be of benefit in further learning. Further description about the C programming language used with the ARX ASURO can be found at: http://www.gnu.org

Note: Descriptions of the commands and functions are also interspersed into the programming lessons when a new topic is covered.

When you don't understand a command, there are three places to look:

(1) The C Programming Essentials in this chapter

(2) The Appendix of this book

(3) C language Reference Manual:
http://www.gnu.org/software/gnu-c-manual/gnu-c-manual.html

29. C Programming Essentials

This section includes some of the information from the ARX ASURO manual along with additional information about the C language to use as a reference while programming. *Additional C language references are included in the Appendix of this book.*

The activity is designed to introduce the primary syntax and language structures that are used in almost every program we will write.

A. Activity – C Programming Essentials

In the activity, we begin with the program code below. We review the lines in the code and add a few commands.

```
#include "asuro.h"
int main(void) //main routine
{
    Init();
    // here we insert our own function blocks
   for (;;)
        {
          SerWrite(,);
         Sleep(255);
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives
With this activity we will learn:

- What is a main routine and how is it defined?
- What are some basic syntax and code structure rules that must be followed?
- What are compiler directives and how are they defined?
- What are variables and how are they defined?
- How is data stored in variables?
- How are numeric variables incremented and decremented?
- How are arithmetic operations performed?

B. Syntax and Code Structure

The ARX ASURO microcontroller CPU executes commands in a C-program step by step from the beginning to the end of the program. Parallel processing is not provided in ARX ASUROs standard processor, so we will have to think in sequential working order when writing programs.

1) Main Routine

There is always a main routine and other functions are called from that routine. The end of the main routine is the end of the program (example: return 0;). There is no special command that designates the end of the program. Here is an example main routine:

```
int main (void)
{
        // insert some code here
 return 0;
}
```

2) Indenting Code

Empty spaces and tabs at the beginning of lines are simply structural reading aids and may be left out. Inset spaces however provide excellent programming structure and are extremely helpful in reading and keeping an overview in extended programs.

3) Ending Commands

In C each command has to be completed with a semicolon “;”, enabling the compiler to assign all elements to the correct command.

4) Grouping Commands

If commands are to be joined in functions, loops or conditional statements, they are grouped by placing them inside a pair of special brackets { }. This allows groups to be nested within groups – especially when creating conditional statements.

Example:

```
#include “asuro.h”
int main(void) {
/* All elements between these brackets belong to one set of commands */
}
```

5) Comments in Programs

If we want to place a comment between programming lines, the set of comment words or lines starts with “/*” and ends with “*/ ”. A single line may also be defined to be a comment by starting the line with “//“. Transforming programming lines into comments is often used to exclude parts of the program from being processed. The compiler will ignore all comments. Correctly inserted comments will never cause problems in program execution.

Example:

```
#include “asuro.h”
int main(void) {
/* This is a comment. All elements between these brackets
     belong to one set of commands  and can go for multiple lines*/
// This single line is also a comment
}
```

6) Compiler Directives

You may have wondered about the first program line #include “asuro.h”. This #include directive defines an inclusion of an external source code into your program and orders the compiler to include the textfile in the compiler sequence. Our include-file “asuro.h” contains some functions, which are needed for robot operations.

Another important directive (amongst others, which are beyond the scope of our introduction to C) is the so called text replacement. This directive is defined by:
#define NAME replacement_text
and is used to define constants in our programs.

Wherever the symbol NAME is found in the source text, it will be replaced automatically by replacement_text. For the NAME following #define the same naming conventions as for variables have to be applied. It is common practice for C-programmers to write the symbols (eg. NAME) at the #define in capital letters.

Example:

```
#include “asuro.h”
#define STARTINGVALUE 33
int main(void)
{
          int i;
          i= STARTINGVALUE; // the value for i is now 33
          return 0;
}
```

Note: Compiler directives are not closed by a semicolon!

7) Variables and Data types

In programming, variables are being used as container elements for data and they may be defined, filled, read or changed in the course of the program. In order to use a variable, it has to be declared at first. In declarations, variables will be assigned to types and they also may be provided with an initial value. Types define which kind of data are to be stored inside the variable (integer numbers, positive integers, real numbers, etc....).

Variables are referenced by names, which have to begin with a letter (underscore "_" will be counted as a letter as well) and may also contain numbers, but none of the other special characters. Capital letters and noncapital letters will be discerned. As an example: x and X reference to different variables. Normally capital letters are being used for constants (which cannot be changed in program flow, e.g. th number PI = 3.1415) and small letters are being used for variables.

a) *Declaring Variables*

Declaration of variables may be provided as global variable outside the main()-function or as a local variable inside the main()-function. Global variables will be valid in the complete program area. Local variables will be valid inside the main()-function or any other function only. They will be valid for the programming code inside the function and they will be invalid outside the function.

Variables are rather useless, until we fill them with data. Assigning data is however rather easy:

```
a=17; // variable a contains a value 17
```

or alternatively in a calculation:

```
a=17+23; // variable a contains a value 40
Speed=a+3; // variable Speed contains a value 43
Speed=Speed*2; // variable Speed contains a value 86
```

Variables may also be declared as a particular data type. See Data Types.

b) *Naming variables*

It is good programming practice to use clear understandable names. The variable name "speed" in this example is self-explainable. Using simple letters as a variable should be restricted as it often results in "unreadable" programming code.

The following example may demonstrate a simple, complete programming source:

```
#include "asuro.h"
int main(void) {
        int i; // i may contain numbers between -32768 and 32767
        char any_token;
        // variable any_token may contain ASCII-symbols or
        // numbers between -128 and 127
        i=3;
        any_token =17+i; // any_token will now be assigned 20
        i=i/2; // Division by 2, will always be rounded down
        // therefore i will now be assigned 1!
return 0;
}
```

c) *Incrementing and decrementing Variables*

A few interesting shortcuts are useful in the programming language C, e.g.:

i=i+1; *may be written as:* i++;

i=i-1; *may be written as:* i--;

d) *Reserved Words*

The following names are reserved words and cannot be used as names for variables:

Auto	default	float	long	sizeof	union
Break	do	for	register	static	unsigned
Case	double	goto	return	struct	void
Char	else	if	short	switch	volatile
Const	enum	int	signed	typedef	while
Continue	extern				

e) *Data Types*

Type	Range of values	Remarks
Char	-128 … +127	a (8 bit long) byte, may store a character from the alphabet
Unsigned char	0 … 255	unsigned characters, may store positive values only.
Int	-32768 .. +32767	two byte values in the range -32768 .. +32767
unsigned int	0 … 65535	positive integers in the range 0 … 65535 float real numbers

Example variable declaration with data types:

```
int c; // Declaring variable c as an int
unsigned int x; // Declaring variable x as an unsigned int
char mult1,mult2; // Defining two variables mult1 and mult2 as char
unsigned char a, b; // Defining two variables a and b as unsigned char
```

8) Arrays

An array is a variable that stores a list. An array is a data structure that lets you store one or more elements consecutively in memory. In C, array elements are indexed beginning at position zero, not one. Declare an array by specifying the data type for its elements, its name, and the number of elements it can store. Here is an example that declares an array that can store ten integers:

```
int my_array[10];
```

You can initialize the elements in an array when you declare it by listing the initializing values, separated by commas, in a set of braces. Here is an example that stores the values 0 thru 4 in an array:
int my_array[5] = { 0, 1, 2, 3, 4 }; There are several other methods of storing data in an array.

You can access the elements of an array by specifying the array name, followed by the element index, enclosed in brackets. Remember that the array elements are numbered starting with zero. Here is an
Example where the number 5 is stored in array position 0:

```
my_array[0] = 5;
```

Here is an example where the second position in my_array is compared to the value 6

```
If my_array[1] == 6
```

In the IR communication program, an array called data is used with the SerRead() function to store keystrokes entered. In this example, the keystrokes are compared to specific characters and then a message is written to screen using SerWrite() function.

```
{
        unsigned char data[] = "012345";
        Init();
        while(1) {
                SerRead(data,5,0); //use data array, 5 values, set blocking
mode
                if (data[0] == ’A’ && data[1] == ’S’ && data[2] == ’U’
                        && data[3] == ’R’ && data[4] == ’O’)
                        SerWrite("HELLO!",6); // Output HELLO
}
```

C. Robot Programming Functions

A few functions have been developed to support ASURO’s programming. These functions are by no means optimal solutions, and for some purposes you may wish to write your own special functions. We will use these functions in the programming activities and they are part of the “asuro.h” file that is included at the beginning of a program.

Note that controlling functions like motor-controls or display functions change a status, which will remain set until changed again. For example, a green LED display will remain green until changed to another color or switched off.

1) Init()

void Init(void)

This function will reset the microprocessor to its initial state and must always be executed at the beginning of a program. If the function is missing, the processor does not even know what to do with its terminals. A simple ARX ASURO program should at least look like:

```
#include “asuro.h”
int main(void) {
        // here we declare some variables
        Init();
        // here we insert our own function blocks
        while(1); // eternal loop
        return 0; // will never be executed
}
```

Why did we insert the eternal loop at the end of the main () -function?

```
        while(1); // eternal loop
```

Normally the main () - function will be closed by return 0; marking the end of the program.

In ARX ASURO some parts of old programs flashed earlier, may still remain in the memory and be executed, resulting in strange effects. To avoid abnormal execution of old program fractions we “catch” the program after execution in an eternal loop. This way we can be sure the program ends in a defined state.

2) StatusLED()

void StatusLED (unsigned char color)

The Status-LED (D12) will be switched OFF or ON. Valid parameter values are
OFF, GREEN, RED or YELLOW

Example:
The Status-LED will be switched ON in red by:

```
        StatusLED (RED);
```

Here is a complete example program:

```
#include „asuro.h“
int main(void) {
        Init();
        StatusLED (YELLOW);
        while(1); // eternal loop
        return 0;
}
```

3) FrontLED()

void FrontLED (unsigned char status)

The Front-LED (D11) will be switched ON or OFF. Valid parameters are ON respectively OFF

Example:
The front-LED will be switched on by:
FrontLED(ON);

4) BackLED()

void BackLED (unsigned char left, unsigned char right)
The Back-LEDs (D15 and D16) will be switched ON or OFF. The first parameter describes the state of the left back-LED (D15) while the second parameter describes the state of the back-LED at the right side (D16). Valid parameters are ON respectively OFF

Example:
The back-LED at the right side (D16) will be switched ON and the back-LED at the left side (D15) will be switched OFF by:
BackLED(OFF,ON);

5) Sleep()

void Sleep (unsigned char time72kHz)
This function will command the processor to wait some time. The waiting period may be defined by a parameter (unsigned char)4., containing a number of maximal 255 and counting cycles from a 72kHz-timer.
Example:
The processor should sleep for ca. 3ms. ==> (0.003s/1) / 72KH = 216.
The function sleep (216); will force the processor to wait for 3ms.
Sleep (216) ;

6) MotorDir()

void MotorDir (unsigned char left_dir, unsigned char right_dir)

This function controls the direction of both engines and should be called before calling speed controls. Valid parameters are FWD (Forward), RWD (Backward), BREAK (Brake or sudden stop, by short circuiting the motors in a transistor bridge) and FREE (Freewheeling).

Example:
The left motor should be moving forward, while the right motor is halted.
MotorDir(FWD,BREAK);

7) MotorSpeed()

void MotorSpeed (unsigned char left_speed, unsigned char right_speed)

This function controls the motor speed for both engines. Maximum speed is 255 (unsigned char).

The motor will start rotating at a value of around 60, depending on mechanical conditions. The parameter value in fact controls motor power and the rotational speed also depends on other factors like friction or inclination in slopes.

As soon as this function has been executed ARX *ASURO will start moving. Sometimes the program however results in unexpected movements and we must take care* ARX *ASURO cannot do any harm to others or itself.*

Example:
The left motor is to move at maximum speed, the right motor not at all. The direction of movement has previously been defined by MotorDir ().
MotorSpeed (255,0);

8) SerWrite()

void SerWrite (unsigned char *data, unsigned char length)

This function outputs data from ARX ASURO by the serial IR-interface at 2400Bit/s, No-parity, 1 StopBit, NoFlowControl. The first parameter contains the reference to the data to be sent, while the second parameter describes the number of characters to be sent.
Example:
A string „Hello how are you?“ should be sent by IR-interface:
SerWrite (“Hello how are you?”,18);

9) SerRead()

void SerRead(unsigned char *data, unsigned char length, unsigned int timeout)

Once you are able to transmit data by IR-interface, you may probably also wish to receive some data. This function will allow you to do so. The first parameter is a pointer to the storage address, where you would like to store the message. The second parameter describes how many characters are expected, while the third parameter describes a timeout period. Timeout is used to prevent eternal waiting periods if less data than expected do arrive. If after a given timeout period no more data arrive, the function will be aborted and the first character in the received string will be replaced by a ‘T’ (=Timeout). If you however define a third parameter ’0’, the function will not abort, but wait, until the last of the expected number of characters has arrived.

Example:

The string „Go Ahead“ should be received and we want to be sure all characters do have arrived at the robot before we continue operation.

```
#include “asuro.h”
int main(void) {
        char data[8]; // allocate storage
        Init();
        SerRead (data,8,0); // Read data
        MotorDir(FWD,FWD);
        MotorSpeed(120,120);
        while(1); // Eternal loop
        return 0;
}
```

10) LineData()

void LineData(unsigned int *data)

This function has been developed to read the light intensity of both phototransistors on the bottom side of ARX ASURO. You will have to define an address pointer to a dual integer location in memory. The function will transfer the AD-converter values of measurement data by both phototransistors.

The first integer value represents the converter value of the left (T9), the second integer value represents the converter value of the right (T10) phototransistor. Maximum intensity (brightness) is referenced by ’1023’, while total darkness is referenced by ’0’. Normally these extreme values will not be found, and in practice measurement values will be found in between.

Example:

Reading the light intensity of both phototransistors (T9, T10)

```
unsigned int data[2]; //Allocate storage
...
LineData(data);
```

data[0] contains the value measured by the left phototransistor (T9)
data[1] contains the value measured by the right phototransistor (T10)

Example:

```
#include “asuro.h” // Line tracing the easiest way
int main(void) {
        unsigned int data[2]; // Allocate storage
        Init();
        FrontLED(ON); // Switch ON line trace illumination
        MotorDir(FWD,FWD); // Both engines go ahead
        while(1){ // Eternal loop, ASURO should follow
                // a line eternally
        LineData(data); // Read brightness data from Phototransistors
        if (data[0]>data[1]) // left brighter than right
```

```
                {MotorSpeed(200,150);} // ... speed up left motor
        else
                {MotorSpeed(150,200);} // ... speed up right motor
        }
        return 0;
}
```

11) OdometrieData()

void OdometrieData(unsigned int *data)

The function scans the reflected light-sensor: Both LED (D13, D14) are activated and the function returns the AD-converter values of the phototransistors (T11, T12). As in the function LineData() a storage area with two integer values must be provided, which will be filled by the function. The first integer value contains the AD-value delivered by the left (T11), the second integer value contains the AD-value delivered by the right (T12) phototransistor. Maximum brightness is referenced by a value ’0’ while absolute darkness is represented by value ’1023’ 5. Normally these extreme values will not be found, and in practice measurement values will be found in between.

Example:

Scanning the reflected light-sensors.

```
unsigned int data[2]; //Allocate memory
..
OdometrieData(data);
```

data[0] containes data from the left phototransistor (T11)

data[1]containes data from the right phototransistor (T12)

Just to prevent misinterpretations: OdometrieData() does not provide the number of revolutions, but the actual illumination at the reflected light-sensors. Discerning bright and dark levels, counting light-dark transitions and counting the number of revolutions is left to be a programmer’s job.

12) PollSwitch()_

unsigned char PollSwitch (void)

This function scans the position of switches (K1-K6) and returns one byte, containing information, which switches have been activated. Switch 1 will set the first bit number 5, switch 2 will set the second bit, ... switch 6 will set bit number 5,

Bit0 (1) -> K6

Bit1 (2) -> K5

Bit2 (4) -> K4

Bit3 (8) -> K3

Bit4 (16) -> K2

Bit5 (32) -> K1

Activating switches 1,3 and 5 will cause the function to return 42 (32+8+2 = 42).
To be sure, the function may be called several times in sequence to provide a “correct” answer.

Capacitor C7 first must be discharged, which may take some time. If you scan the A/D-converter prematurely, some unreliable data may be delivered.

Example:

```
unsigned char taste;
..
switch = PollSwitch();
if (switch>0) {MotorSpeed(0,0);}
```

30. Our First Program

Note: all example programs were copied to your computer when you installed the software, or are available for download with the activities. Do not copy and paste the code from a book into Programmers Notepad, it almost always results in errors that must be fixed.

A. Activity – Our First Program

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity.

We will start the activity with this program code:

```
include "asuro.h" // Hello world in Asuric language
int main (void)
{
        unsigned int i;
        Init();
        while(1) {
                StatusLED(RED); // Status LED turns Red
                for (i = 0; i < 847; i++){
                        Sleep(255);}
                StatusLED(GREEN); // Status LED turns Green
                for (i = 0; i < 847; i++){
                        Sleep(255);}
                SerWrite("Hello World\n\r",14); // Write Hello world
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

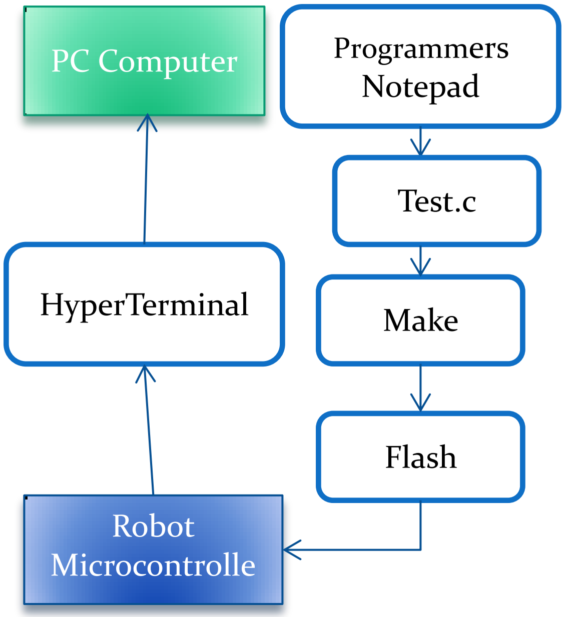
With this activity we will learn:

- What is the overall process that we will follow for all programs?
- How do we write a message from the robot to the PC screen?
- How do we use these ASURO functions?
    - The Init() function
    - The Sleep() function
    - The StatusLED() function
- What are loops and how are they written?
    - For loops
    - While loops

B. Programming Process

With our first program we demonstrate the process that is used for all the programs we will write. We use a software tool called Programmers Notepad to open sample code and write our programs. We save each program with their own name. And then we copy the code and paste it into a program named "test.c" and that is the one that we always compile and download to the robot.

This may seem a bit strange, but it works and saves quite a few steps in the overall process.



We use a command called Make which calls a batch process that compiles the program and creates hex code. Then we use a program called Flash to transfer that hex code to the robot.

When we want to display information about what a program is doing or communicate from the robot to the PC, we use a program called Hyperterminal. It interfaces with the IR communication circuit on the robot.

The hyperterminal program was originally developed to communicate remotely through a modem. So there are some settings that have to be changed (to use IR instead of a modem) and several messages to be ignored each time it is run. This is a bit awkward and hopefully at some time in the future this tool will be replaced with one that is better suited for IR communication. This will become clearer when you actually see the process demonstrated and do it yourself.

C. Copy Code into Test.c

Follow the instructions in the activity to copy code from the program into test.c. Basically, these are the steps:

- Select all the code in first program (Ctrl-A)
- Copy the code from first program (Ctrl-C)
- Switch to the test.c program
- Select the code in test.c (Ctrl-A)
- Paste the copied code, replacing all the code in test.c (Ctrl-V)

D. Make - Compiling a Program

Select Save to Save the test.c program file.

Choose the menu Tools -> make, observing the messages in the display window.

Wait a few seconds until the program finishes the report. Check the status window for a bottom line *Process Exit Code: 0*, indicating the source has been processed correctly by the compiler.

If the compiler displays another message, you will have to trace the error by analyzing the source code. It is a good idea to look at the first line in the source, in which errors have been reported.

The editor displays the line number in the source in the left bottom corner of the display window. After a successful translation you may deliberately alter a word in the program source. Restart the make procedure and check, in which line number the "error" has been reported.

E. Flash – Transfer Program to Robot

Once again, follow the instructions in the activity. Basically, here are the steps for flashing:

- Following an error free compilation, the new program may be flashed. First we connect the IR-Transceiver.
- Then start the Flash program.
- Select the newly created program "test.hex" and the correct COM-interface.
- Place ARX ASURO near the IR-Transceiver
- Press the Programm button and wait until the transfer of the file has been completed. The status line in the display will report a successful completion of the file transfer.
- Switch OFF ARX ASURO and switch ON again.
- Wait a second and check the status-LED to see if it is switching to red.

F. Start the Program on the Robot

As soon as the Flash-program has been executed, the PC will try to communicate with ARX ASURO.

By switching ON the ARX ASURO, the system will be booted, indicated by the Status-LED bicolored lighting for one second. ARX ASURO is checking to see if new software has been prepared to be loaded. If a new program has been found, it will be loaded.

After the program is loaded, it is started by switching the ARX ASURO OFF and ON once again.

G. General Information about Flash and Make

1) Flash Errors

The following errors may occur while flashing:
"c" Checksum Error. ARX ASURO has received some irregular signals. Signals may have been disturbed by other optical sources, such as fluorescent lights, or have been interrupted shortly by movements.

"t" Timeout. The line-of-sight between ARX ASURO and the IR-Transceiver has been interrupted completely.

"v" Verify Error. ARX ASURO wrote invalid data into its Flash-memory. This is a most unusual situation, indicating the non-volatile programming memory (Flash-EPROM) has reached the end of its lifetime, according to specifications after approximately 10.000 programming cycles.

Note: If any of these errors occur, you must repeat the Flash process. Also, there may be times when no errors occur and the program still is not loaded onto the ASURO.

Error correction can be retried ten times. In case of failure the flash-procedure will be aborted.

If "c" Checksum Errors are being indicated regularly while Flashing, you may switch off or dim some lights in the room, especially fluorescent lights.

Remember: Always press the Program-button, before switching ON the ARX *ASURO. Otherwise the download procedure will not be started.*

2) The Make File

The make file in our example program is written in a way that a file with the name test.c will be compiled together with asuro.c (which contains some pre defined functions) and create a new .hex-file. This file can be loaded (flashed) into the robot's memory.

Attention! This means that as long you do not change the make file and you only copy it - you should always name your own program test.c That is why we copy the program code into test.c and then run Make.

3) What is inside a Make File?

When you want to know all about make files (This is not absolutely necessary for operating the ARX ASURO) you can find more background information at http://www.gnu.org/directory/make.html

A makefile controls the compiler and linker process, defining the source code and the methods for compiling and binding, and the output data formats. It looks complicated and so it is. Commercially available tools may be parameterized by mouse clicks, but our system requires creating a makefile by editing, which is relatively easy for the ARX ASURO. Normally we simply copy a working makefile and adapt the command lines. To do so we analyze the makefile in directory `FirstTry` and take a look at the ARX ASURO-related parameters.

If you want to write makefiles from scratch to understand all details, use an Internet search engine to find relevant topics about these themes.

Let's return to our example makefile. In makefiles the #-sign is interpreted as a comment marker:
Anything beyond the # in this line will not be interpreted. The first important command is the MCU (Micro-Controller-Unit) - Name, which is a Atmega8 Processor in ASURO, resulting in: `MCU = atmega8`
We are definitely not allowed to change this line, for it would force the compiler to generate code for another processor type, resulting in strange behavior by the ASURO or no action at all.

The next command controls the output format. All tools for further processing require the Intel-hex-Format (*.hex file))

```
FORMAT = ihex
```

The next lines will be more interesting. By `TARGET = Name` we are defining the name of the resulting output file (*.hex-Data) without file extension. The name is not to be chosen ad lib, as it must be accompanied by a source file (*.c-file). This source normally also defines the `main()`-function.

The next line is especially interesting for experts, as it controls optimizing procedures for the code generator. We may choose between optimizing level 0, 1, 2, 3 and s, in which 0 defines no optimizing at all, and s optimizes for minimum required memory volume. "s" respectively "3" is by no means always the optimal optimizing parameter. Sometimes we need to experiment to find the best option.

In developing a major project (functional self-test including all demo-programs) it usually is a good idea, to split the project into several *.c (C-Code) and/or *.s (Assembler Code) files. In order to let the compiler know, which files are to be included in the project, the sources are included in the variable SRC.

One special TARGET file will be included automatically:

```
SRC = $(TARGET).c
```

All other files will be included in the following lines, simply adding a number of lines:

```
SRC += filename.c
```

or listing all files in a single line. If you need to close a line and continue a new line, you will need to close the preceding line by \ . If you also want to include sources in assembler code, these may be added by:

```
ASRC = filename.S
```

All other commands in the makefile will be skipped to this point. We will simply assume the person responsible for building the makefile example did do a good job, resulting in a good tool. To understand all details please start your favorite search engine and look for "How to write makefiles".

H. Debugging Programs

The process of locating and removing errors in a program is called “debugging”. There are many C programming and debugging techniques that may be covered. However we will limit this to just a few that may come in handy with programming the ASURO. Here are some common issues and debugging procedures to try when your program is not working.

1) Common Compiler Errors

"Nothing is working and ASURO is completely out of control!"
In these situations the statement, the PC or the ASURO faithfully executed the commands we ordered, does not really help solve these problems. Most often we did not work accurately enough. To work efficiently while debugging these errors, we need a guideline. Debugging is rather time-consuming and may easily consume 80% of the total developing time for software.

Only error free compiled programs will be allowed to be flashed into the ASURO memory, otherwise flashing will reload the old, unaltered program into the ASURO.

Generally we always should read the warnings at the compiler sessions. *A lot of problems arise from ignoring compiler warnings.* Errors and warnings are displayed in the compiler-window and will be highlighted.

Messages must be noticed and errors must be repaired. The module (in this case `test.c`) and the line number of the error in the source code will be displayed along with the error description, which of course is to be interpreted correctly. The error list may sometimes be very long and generally it is a good idea to start at the first warning or error message, as the very first bug may be generating many following error messages. Often the problem is not located at the reported line, but a few lines earlier.

Sometimes it is a good idea to show the program to another person and let someone else interpret the source code. It will take some debugging experience to write software, but advanced programmers may experience the "sudden insight" within seconds!

Compile Error: warning: return type of ‘main’ is not ‘int’

The compiler thinks the `main()`-function should return an `int`-value. We solve this by making sure we have a main routine:

```
int main (void)
```

This line will define `main()` as a function returning an int value. This function needs to return a value, so at the end the function we insert a line to eliminate the warning:

```
return 0;
```

We also check to see if we have the beginning and ending curly braces {} in the function.

"Parse Error" before "}" token.

This means someone made a common mistake by forgetting to insert a ";". Insert the semicolon at the end of the prior command and that should resolve the error.

2) The program compiled with no errors but the robot is not working.

Sometimes we have error free-compiled programs running on the robot, but it is reacting quite abnormally. Unfortunately in the ARX ASURO we cannot analyze the program step by step, reading variables or setting interrupts[6].

An easier and simpler method for debugging implies the use of ASURO's indicators, displaying a condition status (if true light a green LED, and a red LED if false). Also it is a good idea to send ASCII-text messages by IR-interface to a terminal program running on your PC, using the function SerWrite().

However this method may not work if the moving ARX ASURO runs out of the transmitter range or if the data transfers slow down the normal program flow.

3) Programming Techniques to Avoid Errors

An experienced programmer will start writing good quality software from the beginning and avoid long debugging phases. Try these Software Engineering tips:

- Start by thinking about the specification of your program, the program structure and the available methods for accomplishing your goal. There is no reason to be ashamed of using a sheet of paper and a pencil for your design.
- Don't produce spaghetti-code, but do split your modules into function blocks, which support the reuse of functions in further projects.
- Test your functions individually to isolate the working blocks from untested sources.
- Immediately write the documentation to your coding using comments. This may seem to be pointless and time-consuming, but it will be well-worth it if you are reading your own code six weeks later: you will not recognize your own coding!

- Use short variable and constant names that mean something instead of cryptic single letter variables.
- Follow programming convention and use lowercase letters for variables and uppercase letters for constants.
- Always check for semicolons and curly braces as they are the most common mistakes made in C programming.

31. IR Communication Program

As you know, to program and communicate with the robot, we use Infrared communication devices. In past activities we have installed and used the Hyperterminal program which sends data to the robot and receives data from the robot.

In the IR communication program we learn more about how to display data on the PC screen using the Hyperterminal. We also learn how to type information on the PC keyboard and read that information on the ARX ASURO robot. This will allow us to control the robot.

A. Activity – IR Communication

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity.

We will start the activity with this program code:

```
include "asuro.h" // IR Communication 1
int main(void)
{
        unsigned int i;
        Init();
        while(1) {
                SerWrite(“John Doe”,9); // output a name
                for (i = 0; i < 282; i++)
                        Sleep(255); // wait
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

- How do we write a message from the robot to the PC screen?
- How do we obtain information from the keyboard?
- What are arrays and how are they used?
- How do we use these ASURO functions?
    - The SerWrite() function
    - The SerRead() function
    - The Sleep() function
    - The StatusLED() function
- Review - What are loops and how are they written?
    - For loops
    - While loops

32. Controlling LEDs Program

In the Hello World program we flashed one LED. In this activity we control multiple LEDs, with timing to turn them on and off and create a flashing effect. We also change the color of the bicolor LED. Then we learn to use the PC keyboard to turn particular LEDs on or off. After completing these programs, we will be able to confidently control any of the robot's LEDs using either timing or PC control.

A. Activity – Controlling LEDs

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity. We will start the activity with this program code:

```
#include "asuro.h"
int main (void)
{
        unsigned int i;
        Init();
        while(1) {
                // Status LED Turns Red
                for (i = 0; i < 282; i++){
                        Sleep(255);}
                // Status LED Turns Green
                for (i = 0; i < 282; i++){
                        Sleep(255);}
                // Status LED Turns Off
                // bottom LED Turns on
                for (i = 0; i < 282; i++){
                        Sleep(255);}
                // bottom LED Turns Off
                // Back left LED Turns on
                for (i = 0; i < 282; i++){
                        Sleep(255);}
                // Back right LED Turns on
                for (i = 0; i < 282; i++){
                        Sleep(255);}
                // Back LED Turns Off
                // Status LED Turns Green
                for (i = 0; i < 282; i++){
                        Sleep(255);}
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

- How do we turn on and off the LEDs?
- How do we make the LEDs flash?
- How do we change the color of the Bi-color LED?
- How do we use these ASURO functions?
    - The FrontLED(), function
    - The BackLED(), function
    - The Sleep() function
    - The StatusLED() function
- How do we increment and decrement variables?
- How do we define constant values and use them in a program?
- How can we use the keyboard to control turning LEDs on and off?
- Review - What are loops and how are they written?
    - For loops
    - While loops

33. Driving – Motor Control Program

We are finally ready to take control of the motors and get the robot to move. In this activity, we write a series of programs to do just that. We begin by learning how turn the motors on and off and then progress to more advanced moves.

We control the direction and speed of the right motor and then the left motor with the MotorDir() and MotorSpeed() functions. Then we calibrate the motor speed so the robot drives in a straight line both forward and backward. We then learn to make a right and left turn.

Once we have mastered turning, we program the robot to drive in a rectangular pattern. Finally, we create a program to control the robot using the PC keyboard.

A. Activity – Driving Controlling the Motors

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity. We will start the activity with this program code:

```
#include "asuro.h"
/* ----- Forward drive ---- */
void PCFront(void)
{
        // Set motors to drive Forward
        // Set motor speed to drive in a straight line
}
/* ----- Reverse drive ---- */
void PCBack(void)
{
        // Set motors to drive Reverse
        // Set motor speed to drive in a straight line
}
/* ----- Left Turn ---- */
void PCLeft (void)
{
        // Set motors to drive Forward
        // Set left motor speed to 255
}
/* -----Right Turn ---- */
void PCRight (void)
{
        // Set motors to drive Forward
        // Set right motor speed to 255
}
/* -----Stop ---- */
void PCStop(void)
{
        // Set Motors to BREAK
        //Set Motor Speed to zero
}
/* ----- Main ---- */
int main (void)
{
// here we insert code to call driving functions
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

- How do we turn on and off the motors?
- How do we control the speed of the motors?
- How do we turn left and right?
- How do we stop?
- How do we use these ASURO functions?
    - The MotorDir() function
    - The MotorSpeed() function
    - The SerWrite() function
    - The SerRead() function
- How do we write our own functions?
- How do we write a switch statement to compare multiple cases?

34. Line Following Program

By this time, we have mastered making the robot move under programmatic control. Next, in the Line Following Activity we make the robot find and follow a black line on a light colored floor.

We first learn to drive forward until it finds a line and then stop. We call this "Stopping at the Abyss". Then we program the robot to follow a line in an oval shaped path.

In these programs we use the LineData() function in our LineLeft() and LineRight() functions. We also turn on LEDs to indicate which direction the robot is turning. We will also use the own functions for moving forward, reverse, and turning left and right from the last activities.

A. Activity – Line Following

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity. We will start the activity with this program code:

```
#include "asuro.h"// Reading sensor signals
int main(void)
{
        unsigned int lineData[2];
        Init();
        while(1) {
                LineData(lineData);
                if (lineData[0] > lineData[1])
                        StatusLED(GREEN);// Status LED turns Green
                else
                        StatusLED(RED); // Status LED turns Red
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

- How do we read the photo sensors for line following?
- How do we control the motors to stop when we find a line?
- How do we use these ASURO functions?
    - The LineData () function
    - The FrontLED () function
    - The BackLED () function
    - The StatusLED() function
    - The MotorDir () function
    - The MotorSpeed () function

- How do we write our own functions?
- How do we use arrays to store and retrieve data?
- How do we use Hex numbers in commands?
- How do we convert data from one format to another?
- How do we display Hex data for troubleshooting purposes?
- How do we create a do loop?

35. Object Detection and Avoidance Program

After mastering robot movement under program control and line following, next we program the robot to detect when it collides with or bumps into an object. Our objective will be to sense a collision and then take action to be able to avoid or go around the object detected.

Our program will be based upon sensing the 6 collision detection switches on the front of the robot. We will use the PollSwitch() function and also create a SwitchTest() function. We will also use the driving functions we wrote in a prior activity.

A. Activity – Object Detection

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity. We will start the activity with this program code:

```
include "asuro.h"
/* ----- Switch Recognition ---- */
void SwitchTest(void)
{
        unsigned char sw;
        sw = PollSwitch();
        StatusLED(OFF);
        FrontLED(OFF);
        BackLED(OFF,OFF);
        /* K6 Switch */
        if (sw & 0x01) {
        SerWrite("K6 Switch\n\r",12);
        }

        /* K5 Switch */
        if (sw & 0x02) {
        SerWrite("K5 Switch\n\r",12);
        }
/* K4 Switch */
        if (sw & 0x04){
        SerWrite("K4 Switch\n\r",12);
        }

        /* K3 Switch */
        if (sw & 0x08){
        SerWrite("K3 Switch\n\r",12);
        }

        /* K2 Switch */
        if (sw & 0x10){
        SerWrite("K2 Switch\n\r",12);
        }
        /* K1 Switch */
        if (sw & 0x20){
        SerWrite("K1 Switch\n\r",12);
        }

}
/* ----- Main ---- */
int main(void)
{
        unsigned char sw;
        Init();
        while(1){
                sw = PollSwitch();
                if (sw > 0){
                        SwitchTest();
                }
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

- How do we know when a switch has been pressed?
- How can we distinguish one switch press from another?
- How do we use these ASURO functions?
    - The PollSwitch () function
    - The FrontLED () function
    - The BackLED () function
    - The StatusLED() function
    - The SerWrite() function

- How do we write and call our own functions?
- How do we use arrays to store and retrieve data?
- How do we use Hex numbers in commands?
- How do we convert data from one format to another?
- How do we display Hex data for troubleshooting purposes?

36. More Control with Encoders Program

The Odometer or Encoder program is one of our most advanced programming activities. In it we obtain data from the encoder using the A/D converter with the OdometrieData() function.

With the encoder data we can calculate the robot's speed and distance traveled. We may then use this information to travel a predefined path and set the speed and distance traveled in a particular direction.

As an advanced activity, we can drive the robot in a straight line, turn left and right, and make geometric shapes using set speed and distances with dead reckoning. Dead Reckoning is the process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course. Note: dead reckoning is subject to cumulative errors.

A. Activity – Encoders

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity. We will start the activity with this program code:

```
include "asuro.h"
/* ---------- Odometrie Sensor Test ------------- */
/* Left Sensor -> Status LED GREEN ON when Light on
   Left Phototransistor bright enough            */
/* Right Sensor -> Status LED RED ON when Light on
   Right Phototransistor bright enough           */
void OdometrieTest(void)
{
        unsigned int data[2];
        OdometrieData(data);
        StatusLED(OFF);
        if (data[0] < 512)
                StatusLED(GREEN);
        if (data[1] < 512)
                StatusLED(RED);
}
/* END Odometrie Sensor Test ------------------- */
/* ----- Main ---- */
int main(void)
{
        // Insert code to run OdometrieTest() and then check the sensor data

// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

- How do we obtain data from the odometer?
- How do we get data for both the right and left wheels?
- What are the valid values for the data – how do we know when a dark or light line has been sensed?
- How can we use data from the odometer to calculate the distance moved?
- How can we use data from the odometer to calculate speed?
- How do we use these ASURO functions?
    - The OdometrieData () function
    - The FrontLED () function
    - The BackLED () function
    - The StatusLED() function
    - The SerWrite() function

- How do we write and call our own functions?
- How do we use arrays to store and retrieve data?
- How do we use Hex numbers in commands?
- How do we convert data from one format to another?
- How do we display Hex data for troubleshooting purposes?

37. Reading Battery Voltage Program

Reading Battery Voltage is also an advanced programming activity. In this activity we use the A/D converter combined with a voltage divider to monitor the voltage of the battery. We then display the voltage to the hyperterminal. The data is in binary format and some use of registers and bit manipulation techniques are used to extract it, which also use pointers to memory locations.

After learning to read and display the voltage, we put the robot up on jacks and run the motors for a predefined period of time and monitor the voltage readings. We then calculate and display the drop in voltage over time.

A. Activity – Using A-D Converter

Follow the instructions in the Activity to start Programmers Notepad and open the program for this activity. We will start the activity with this program code:

```
include "asuro.h" // Reading the battery supply voltage
#define R1 12000.0
#define R2 10000.0
int main (void)
{
        unsigned int value;
        float Uout, Uin;
        unsigned char output[9];
        Init();
        ADMUX = (1 << REFS1) | (1 << REFS0) | (1 << MUX0) | (1 << MUX2);
                // internal reference with external capacitor
        while(1) {
                ADCSRA |= (1 << ADSC); // start conversion
                while (!(ADCSRA & (1 << ADIF))); // wait for conversion complete
                ADCSRA |= (1 << ADIF);
                value = ADCL + (ADCH << 8);
                Uout = (2.56 / 1024.0) * value;
                Uin = ((R1 + R2) * Uout) / R2;
                //Truncate all minor digits in Ue and convert start conversion
                // into ASCII symbols
                output[0] = (int)Uin + 0x30;
                // Insert the first digit in front of the comma
                Uin = (Uin - (int)Uin) * 10.0;
                output[1] = ','; // Voltage number is only one digit
                // Truncate all minor digits in Uin and convert
                // the first minor digit into ASCII symbol = 1
                output[2] = (int)Uin + 0x30;
                // Insert the second digit behind the comma
                Uin = (Uin - (int)Uin) * 10.0;
                // Truncate all minor digits in Uin and convert
                // the second minor digit into ASCII symbol
                output[3] = (int)Uin + 0x30;
                output [4] = '[';
                output [5] = 'V';
                output [6] = ']';
                output [7] = '\n'; // new line
                output [8] = '\r'; // return
                SerWrite(output,9); // generate output
        }
// here we end the main routine
while(1); // eternal loop
return 0; // will never be executed
}
```

Objectives

With this activity we will learn:

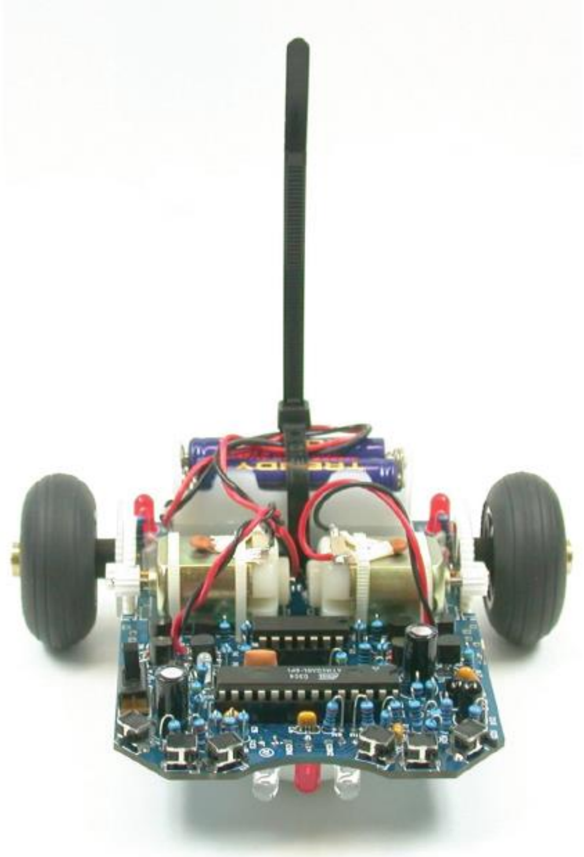
- How do we obtain data from the A/D Converter?
- How do we interpret the data we receive from different registers?
- How can we use bit manipulation techniques to parse the data?
- What are the valid values for the data?
- How can we use the data to calculate the battery voltage?
- How do we use these ASURO functions?
    - The OdometrieData () function
    - The FrontLED () function
    - The BackLED () function
    - The StatusLED() function
    - The SerWrite() function

- How do we write and call our own functions?
- How do we use arrays to store and retrieve data?
- How do we use floating numbers in commands?
- How do we convert data from one format to another?

SECTION 5: Competitions

Classroom Competitions and having fun with the ARX ASURO Robot

- *Competitions Overview*
- *Line Following Competition*
- *Collision Detection Competition*
- *Other Competitions*



Competitions Overview

After learning to program the robot, in this section you are given an opportunity to compete with other students. The competitions allow you to test and demonstrate your programming skills.
The programming activities provide the essential code needed for creating programs for the competitions, but you will be on your own to pull everything together and add logic to create a program for each competition. The competitions include line following and object detection and avoidance as well as others.

Competitions are not only challenging but fun and allow you to express your creativity as well as your competitive nature. These competitions will also prepare you for other robot competitions.
You may discover some hidden abilities and talents as you prepare for competitions and compete with your ARX ASURO robot.

This section provides ideas for a number of competitions of varying levels of difficulty. These competitions require that students program the ARX ASURO robot to meet the specific rules and objectives of the competition. Most competitions involve a completion of a series of maneuvers that are to be completed in the least amount of time. *Each instructor will decide which competitions to include in a specific class and when they will be held.*

General Instruction for All Competitions

In many competitions of this type, students must weigh the pros and cons of speed versus accuracy. Moving too fast may cause the robot to lose its way. Conversely moving too slowly may cause the robot to lose because it took too long to complete the objectives.

Students will receive a list of objectives and rules as well as sample drawings for the course. There is a time limit specified for modifying programs once the real course is seen on the day of the competition.

The method of starting the robots for the competition will be defined by the instructor or judges. In some cases the power button will be turned on, in some cases an IR remote may be used, and in some cases keyboard commands may be used. Where the robot is placed (or held) during the start is also defined.

Note: Besides the points awarded or deducted for the competition rules, additional points may be awarded or deducted for sporty conduct, helping others, and following the competition rules established by the instructor or judges.

38. Line Following Competitions

One of the things that the ARX ASURO is really good at is line following. But you may discover that there are a few tweaks that you can make to the program code or to the hardware that might give your ARX ASURO an advantage over others.

There are three competitions defined here. Instructors and Judges may decide to include any or all of these competitions.

- **Amazing Abyss** - Find the lines to go through a maze
- **Follow as Fast as You Can** - Rapidly following a line in a path
- **Choose Your Path Wisely** - Following a line and choosing a path

A. Amazing Abyss

Compete against the clock and other competitors to complete a course through a maze where the boundaries of the maze are defined by taped lines.

1) Competition Objectives and Rules

- The robot may not cross a line. Only the front LED and photo transistors may cross the line. The robot must back up and turn to avoid the line.
- Wheels are not allowed to touch the line, while driving forward.
- The robot who completes navigating the maze in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape. The robot must use the line following sensors to detect the line instead of using an exact distance.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.
- Points are awarded for each line that is detected and the robot backs up and avoids crossing the line.
- Points are deducted for each line where the robot wheels cross the line.
- Points are deducted if the robot gets lost and cannot find the next line.

B. Follow As Fast as You Can

Compete against the clock and other competitors to follow a complex curved and straight line course. The robot follows one single dark line on a white surface.

1) Competition Objectives and Rules

- The robot must follow the line throughout each section of the course.
- Robots which lose the line after 5 seconds may be picked up and put back on the line.
- The robot who completes navigating the maze in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape. The robot must use the line following sensors to detect the line instead of using an exact distance.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.
- Points are awarded for each section of the course completed successfully.
- Points are deducted when the robot loses the line for more than 2 seconds.
- Additional points are deducted if the student has to pick the robot up and put it back on the line.

C. Choose Your Path Wisely

Compete against the clock and other competitors to follow a complex curved and straight line course. The robot follows one single dark line on a white surface. There are junctions in the path, and the robot may choose to continue forward or make a 90 degree turn at the junction. Each junction may indicate a shorter path to follow or a longer one. Also the junctions may be left or right turns. The student will not know until the day of the competition.

1) Competition Objectives and Rules

- The robot must follow the line throughout each section of the course.
- Robots which lose the line after 5 seconds may be picked up and put back on the line.
- The robot who completes navigating the maze in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape. The robot must use the line following sensors to detect the line instead of using an exact distance.
- The paths and alternate routes may also change in the drawing used for the competition.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.
- Points are awarded for each section of the course completed successfully.
- Points are deducted when the robot loses the line for more than 2 seconds.
- Additional points are deducted if the student has to pick the robot up and put it back on the line.
- Additional points are added if the robot chooses all the shortest paths.

39. Collision Detection Competitions

Another thing that the ARX ASURO can do is detect and avoid obstacles. But you may discover that there are a few tweaks that you can make to the program code or to the hardware that might give your ARX ASURO an advantage over others.

There are three competitions defined here. Instructors and Judges may decide to include any or all of these competitions.

- **Boxed In** - Follow a course through a maze of boxes
- **Remember Me** - Complete the box maze, learn it, and do it again better
- **Avoid It** - Follow a course through a maze of lines with obstacles blocking the path

A. Boxed In

Compete against the clock and other competitors to follow a course through a maze made of boxes. The robot must avoid pushing or knocking down a box. If a box is detected, the robot backs up to avoid it, turns, and continues thru the maze.

1) Competition Objectives and Rules

- The robot must follow the maze throughout each section of the course.
- Robots which lose the maze after 5 seconds may be picked up and put back in the course.
- The robot who completes navigating the maze in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape. The robot must use the collision detection sensors to detect the boxes instead of using an exact distance.
- The paths and alternate routes may also change in the drawing used for the competition.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.

- Points are awarded for each section of the course completed successfully.
- Points are deducted when a box is pushed or knocked down.
- Points are deducted when the robot loses the maze for more than 2 seconds.
- Additional points are deducted if the student has to pick the robot up and put it back in the maze.
- Additional points are added if the robot chooses all the shortest paths.

B. Remember Me

Compete against the clock and other competitors to follow a course through a maze made of boxes. The robot must avoid pushing or knocking down a box. If a box is detected, the robot backs up to avoid it, turns, and continues thru the maze. The robot learns the maze as it goes thru.   After completion of the first time thru, the robot is picked up and put back at the starting point. The objective is to learn the maze and make the fastest time the second time thru the maze without pushing or knocking down a box.

1) Competition Objectives and Rules

- The robot must follow the maze throughout each section of the course.
- Robots which lose the maze after 5 seconds may be picked up and put back in the course.
- The robot who completes navigating the maze the second time thru in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape. The robot must use the collision detection sensors to detect the boxes instead of using an exact distance.
- The second time thru the maze, the robot does not have to use collision detection, but may use encoders and distances recorded from the first time thru.
- The paths and alternate routes may also change in the drawing used for the competition.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time the second time thru.
- Points are awarded for each section of the course completed successfully.
- Points are deducted when a box is pushed or knocked down.
- Points are deducted when the robot loses the maze for more than 2 seconds.

- Additional points are deducted if the student has to pick the robot up and put it back in the maze.
- Additional points are added if the robot chooses all the shortest paths

C. Avoid It

Compete against the clock and other competitors to follow a complex curved and straight line course. The robot follows one single dark line on a white surface. There are two or three soda cans (or other obstacles) placed at random locations on the dark line in this course. If the robot detects an obstacle it is to back up and go around it.

1) Competition Objectives and Rules

- The robot must follow the line throughout each section of the course.
- Robots which lose the line after 5 seconds may be picked up and put back on the line.
- The robot who completes navigating the maze in the least amount of time, avoiding the most cans, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape. The robot must use the line following sensors to detect the line instead of using an exact distance.
- The robot uses sensor switches to detect cans. It must back up and go around the can and find the line again to continue on the course.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.
- Points are awarded for each section of the course completed successfully.
- Points are deducted when the robot loses the line for more than 2 seconds.
- Points are deducted if a can is pushed or knocked down.
- Additional points are deducted if the student has to pick the robot up and put it back on the line.

40. Other Competitions

Relay races, making turns, drawing, and other fun activities are all possible with the ARX ASURO Robot. In these competitions, stretch your imagination and engineering skills to compete. You may also work with a team or a class to design your own competition.

Three competitions are defined here:

- **Drag Race** - Compete for speed on a straight track
- **Turn Right and Left** - Complete a course with lots of turns
- **Keep it Symmetrical** – Program robot to draw shapes with a pen

A. Drag Race

Compete against the clock and other competitors to travel in a straight line as fast as the robot can until it gets to the finish line, and then stop. The only lines are the beginning and ending perpendicular lines of the race.

1) Competition Objectives and Rules

- The robot must start at the designated spot just before the start line and stop just after the finish line, and not go outside the boundary of the track.
- The robot who completes the race in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape.
- The robot must only move forward in a straight path, and must detect both the start and finish lines. The robot indicates it detected the lines by flashing both rear LEDs.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.
- Points are awarded for detecting the start and end finish lines – as indicated by flashing both rear LEDs.
- Points are deducted when the robot veers off and does not go straight.
- Points are deducted if the robot does not stop after crossing the finish line.

- Additional points are deducted if the student has to pick the robot up and put it back in the course.

B. Turn Right and Left

A number of lines are defined using tape. A course is laid out such that at locating each tape, the robot must stop and first execute a 90 degree Right turn and then proceed in a straight line till the next line is reached. At that line it must make a 90 degree Left turn and proceed in a straight line until it reaches the next line. At that line it must stop and execute a 90 degree right turn. The direction of the turns are indicated on the track.

This Right, Left turn sequence is repeated until the robot reaches the finish line, at which time it must stop. The finish line is defined by two lines perpendicular to the direction of travel (very close together within a ¼ inch of each other). It is important to execute 90 degree turns as an error can throw the robot off course and cause it to miss the next line. Errors may also accumulate as the course progresses. The shorter the lines are, and the more distance between them, the harder the course.

1) Competition Objectives and Rules

- The robot must start at the designated spot just before the start line and stop just after the finish line, and not go outside the boundary of the course.
- The robot must make the designated left and right 90 degree turns throughout the course.
- The robot who completes the race in the least amount of time, with the most points, wins.

2) Competition Parts and Prep Work

- A drawing may be provided to use to prepare for the competition. However, be aware that the judges may alter the drawing dimensions or shape or the turns to be made.
- The robot must only move forward in a straight path, and must detect both the start and finish lines. The robot indicates it detected the lines by flashing both rear LEDs.
- The robot must make 90 degree turns with as little error as possible.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.

4) Competing

- Robots try to complete the course in the fastest time.

- Points are awarded for detecting the start and end finish lines – as indicated by flashing both rear LEDs.
- Points are deducted when the robot veers off the course.
- Points are deducted when the correct left or right turn are not made.
- Points are deducted if the robot does not stop after crossing the finish line.
- Additional points are deducted if the student has to pick the robot up and put it back in the course.

C. Keep it Symmetrical

The robots draw a symmetrical geometric figure. Figures are judged based upon complexity and appeal. Drawings are completed by attaching marker pen(s) to the rear of the robot. All robots are given one sheet of paper the same size which is taped down to the floor.

1) Competition Objectives and Rules

- The robot must complete a drawing. It flashes all lights when it begins the drawing and when it ends the drawing.
- The robot can move in any direction as it draws.
- The pen(s) is attached to the rear of the robot.
- The robot who completes the best drawing, with the most points, wins.

2) Competition Parts and Prep Work

- The drawing paper size is indicated to prepare for the competition.
- The robot indicates it is starting and ending the drawing by flashing all LEDs.
- The amount of time allocated for writing programs and preparing for the competition will be set by the judges.

3) Programs

- Write your own C program using code segments and functions from any of the program examples.
- Program code is to be turned into the judges prior to beginning the competition (either printed or electronic).
- Include your name and the name of the program as the first line of comments in the program.
- A sample of the drawing to be completed is also submitted to the judges with the program code ahead of time.

4) Competing

- Robots try to create the most attractive drawing. Drawings are judged on complexity and artistic appeal.
- Points are awarded for flashing LEDs when starting and finishing.
- Points are awarded for drawing complexity.
- Points are deducted if the robot does not stop after finishing, or if it does not complete the drawing.
- Points are deducted if the student has to pick the robot up and put it back on the paper.
- Points are deducted if the pen comes loose and has to be reattached during the drawing.

D. Make Your Own Competition or Programming Challenge

Here are some ideas for additional programming challenges to tackle, and perhaps to convert into a competition.

- Driving at a constant speed and dead reckoning navigation
- Driving curves
- Driving in circles
- Drawing lines and other shapes with an attached pen
- Hitting objects or knocking robots out of an arena
- Big Brother: A herd of ASUROs will be controlled by a central PC system to investigate its surroundings, alternatively including a visual pattern. (The line-tracing LED may also be positioned on the top side).
- Autonomous behavior: autonomously investigating the surroundings.

These require the addition of the extension board and other add-ons which may be available for the ARX ASURO:

- Addition of a CCD camera, providing high-quality video data transfer or directly reading video signals into the ARX ASURO processor at a rate of 32x16 pixel and grey scales or alternatively using a separate additional video processor.
- Microphones for speech recognition or sound detection
- Interfacing with WIFI from a cell phone and control the robot from a cell phone Ap.
- E-field or H-field detection and ranging to detect and follow an electromagnetic trace.
- Pheromone tracing: a leading ARX ASURO is marking a trace and others will follow the trace. This requires at least two ARX ASUROs. Using a Phenolphthalein felt-pen and optical sensors (applying non-red LEDs) or applying real odor sensors you may activate the odor sensor system.
- Redesigning the Infrared communication module for 433- or 868MHz RF transmission for extended distance communication.
- Electromagnetic pulling of objects instead of pushing objects
- Additionally using a mechanical gripper
- Radar detection system for obstacles
- Laser sensors instead of collision detectors
- Heat sensors for tracing heat sources
- Display system providing a few buttons
- Two ARX ASURO systems detecting each other by optical sensors
- Magnetic compass
- Balancing at the rear wheels

41. Appendix

Additional information to use for reference purposes is located here.

A. FAQs - Questions and Answers

1. How do I switch to the demo mode?

ARX ASURO must be completely intact and ready to go. Hardware problems may cause any sort of difficulties! As a precondition all demo programs must be loaded to the processor, which is the initial stage of the processor as shipped in the kit. Otherwise the file SelfTest.hex will have to be transferred from the CD into the ARX ASURO processor memory. To start the demo program complete the following steps:

Activate ARX ASURO and turn the left wheel immediately. The status LED will be activated and not change to red, indicating the self-test starting phase. The buttons may be used to start:
Button 1: Line tracing
Button 2: Odometrics demo
Button 3: PC remote control

2. **Demo Programs have disappeared**

Loading a user-written program into ARX ASURO will remove all former programs from the processors memory including the demo programs and the self-test. In order to reload these programs reload the file SelfTest.hex from the CD into the ASURO processor memory.

3. The Status LED starts flashing at activating ARX **ASURO.**

A green-orange flashing Status LED at activating ARX ASURO indicates a low battery voltage and requires charging or replacement the accumulator/battery. The battery may not contain enough energy to reliably flash the processor.

4. I bought a new processor and the component is not working.

All ARX ASURO processors have been preprogrammed. There is a reserved and protected area in ASUROs memory for the bootstrap loader, handling communications for the flash program and containing all functions to allow ISP (In System Programming). If you need the bootloader you should order the new processor from your ARX ASURO dealer.

5. Flashing does not work

Did you notice repeated checksum errors? Sometimes you will need to switch off the lights in the room or to place ARX ASURO into a darker area, especially if you use fluorescent lights (which by the way do radiate a lot of infrared light in the ARX ASURO spectrum). Of course you may also trim the RS232 Transceiver.

6. PollSwitch() causes problems

This problem may occur, especially while activating switches with high resistor values. The charged capacitor C7 will have to be discharged first, taking some time. If the A/D converter starts an early sample a measurement value may occur ad lib.

To eliminate these factors it may be necessary to repeat the PollSwitch() Function up to three times. It may also happen that the motors are causing interference similar to activating switch.

B. C Programming Reference

1) Operators

a) Arithmetic operators

Operator	Purpose	Examples
+	Addition	x = 5 + 3; y = 10.23 + 37.332; a = b + PI
-	Subtraction	x = 5 – 3; y = 3.45 – 2.34; a = b - PI
*	Multiplication	x = 5 * 2; y = 3.5 * 5.655; a = b * PI
/	Division	x = 5 / 2; y = 2.5 / 7.5; a = b / PI

b) Assignment operators

+=	Adds the two operands together, and then assign the result of the addition to the left operand.
-=	Subtract the right operand from the left operand, and then assign the result of the subtraction to the left operand.
*=	Multiply the two operands together, and then assign the result of the multiplication to the left operand.
/=	Divide the left operand by the right operand, and assign the result of the division to the left operand.
%=	Perform modular division on the two operands, and assign the result of the division to the left operand.
<<=	Perform a left shift operation on the left operand, shifting by the number of bits specified by the right operand, and assign the result of the shift to the left operand.
>>=	Perform a right shift operation on the left operand, shifting by the number of bits specified by the right operand, and assign the result of the shift to the left operand.
&=	Perform a bitwise conjunction operation on the two operands, and assign the result of the operation to the left operand.
^=	Performs a bitwise exclusive disjunction operation on the two operands, and assign the result of the operation to the left operand.
|=	Performs a bitwise inclusive disjunction operation on the two operands, and assign the result of the operation to the left operand.

For example, x+=y is the same as x=x+y

c) *Bit Shifting Operators*

In C, you can only use these operators with operands of an integer (or character) type, and you should only use the bitwise negation operator with unsigned integer type

Operator	Explanation	Example
<<	Use to shift its first operand's bits to the left. The second operand denotes the number of bit places to shift. New bits added on the right side will all be 0.	x = 47; /* 47 is 00101111 in binary. */ x << 1; /* 00101111 << 1 is 01011110. */
>>	Shift its first operand's bits to the right. Bits shifted off the right side are discarded. New bits added on the left side are usually 0,	x = 47; /* 47 is 00101111 in binary. */ x >> 1; /* 00101111 >> 1 is 00010111. */
&	Biwise conjunction examines each bit in its two operands, and when two corresponding bits are both 1, the resulting bit is 1. All other resulting bits are 0.	11001001 & 10011011 = 10001001
|	Bitwise inclusive disjunction examines each bit in its two operands, and when two corresponding bits are both 0, the resulting bit is 0.	11001001 | 10011011 = 11011011
^	Bitwise exclusive disjunction examines each bit in its two operands, and when two corresponding bits are different, the resulting bit is 1. All other resulting bits are 0.	11001001 ^ 10011011 = 01010000
~	Bitwise negation reverses each bit in its operand.	~11001001 = 00110110

Examples using variables:

```
unsigned int foo = 42;
unsigned int bar = 57;
unsigned int x;
x = foo & bar;
x = foo | bar;
x = foo ^ bar;
x = ~foo;
```

2) Conditions

Sometimes we would like to execute commands under certain conditions. These conditional sequences are called control structures. The simplest of these control structures is an "if-else" sequence.

In "C" the correct syntax will be as follows:

```
if (Condition)
    Command_block_1
else
    Command_block_2
```

The program will check the value of the condition between the brackets. If the condition is true (which implies any value except zero), the program will execute Command_block_1 and otherwise the optional Command_block_2.

If the program should be able to choose one option out of several alternatives, you may use several "else if"-constructs.

```
if (Condition1)
    Command_block_1
else if (Condition2)
    Command_block_2
else if (Condition3)
    Command_block_3
else if (Condition4)
    Command_block_4
else
    Command_block_5
```

Example:

```
#include "asuro.h"
int main(void) {
    while (1) {
            if (PollSwitch()>0) {StatusLED(RED);}
            else {StatusLED(GREEN);}
            }
}
```

If one of the collision detector switches has been activated, the status-LED will be switched on red, otherwise in green.
In the C-language a value "1" represents true and "0" represents false.

The conditional statement:

```
if (0) {StatusLED(RED);}
```

of course implies the command StatusLED(RED) will never be executed.

3) Comparison Operators

The following conditional structures may be used in comparisons:

Operator	Explanation
==	comparison for equal
!=	comparison for unequal
<	comparison for less
>	comparison for greater
<=	comparison for less or equal
>=	comparison for greater or equal

4) Logical Operators

AND: The logical conjunction operator && tests if two expressions are both true. If the first expression is false, then the second expression is not evaluated. Here is an example which tests to see if x is 5 AND y is 10.

```
if ((x == 5) && (y == 10))
        SerWrite (``x is 5 AND y is 10",20 );
```

OR: The logical conjunction operator || tests if at least one of two expressions it true. If the first expression is true, then the second expression is not evaluated. Here is an example which test to see if x is 5 OR y is 10.

```
if ((x == 5) || (y == 10))
        SerWrite (``x is 5 OR y is 10",18 );
```

NOT: You can prepend a logical expression with a negation operator ! to flip the truth value. Here is an example which tests to see if x is NOT equal to 5

```
if (!(x == 5))
        SerWrite (``x is NOT 5", 11);
```

Operator	Explanation
&&	AND - logical conjunction operator && tests if two expressions are both true
||	OR - logical conjunction operator || tests if at least one of two expressions it true
!	NOT - prepend a logical expression with a negation operator ! to flip the truth value

5) Loops

Loops are used to repeat command execution.

a) *While Loop*

In a “while”-loop a condition is checked every time the loop is passed. If the condition is true the command block will be executed. The condition is checked again until it turns to false. At a false condition the program is continued at the first command following the condition block.

```
while( Condition)
Command block
```

Example:

```
#include “asuro.h”
int main(void) {
        MotorDir(FWD,FWD); // Both engines running forward
        MotorSpeed(120,120); // Both engines running at around half speed
        StatusLED(GREEN); // Turn on Status-LED green
        while (PollSwitch()==0) { // As long as there is no collision
                SerWrite(“All OK!\n”,10); // … Feeling groovy ….
        }
        MotorSpeed(0,0); // Collision! Stop immediatedly!
        StatusLED(RED); // Turn on Status-LED red
        while (1) {
                SerWrite(“Ouch!\n”,5); // start crying!
        }
}
```

b) *Do Loop*

The do statement is a loop with an exit test at the end of the loop. Here is the general form of the do statement:

```
do
        statement
while (test);
```

The do statement first executes statement. After that, it evaluates test. If test is true, then statement is executed again. Statement continues to execute repeatedly as long as test is true after each execution of statement.

This example prints the integers from zero through nine:

```
int x = 0;
do
        printf ("%d ", x++);
while (x < 10);
```

Here is an example used in line following:

```
do {
        LineData(lineData);
        if ((lineData[0] < STOP) && (lineData[1] < STOP)) {
```

```
        MotorSpeed(0,0); // Set Motor speed to Zero
        running = FALSE;
  }
} while (running);
```

Notice a variable is used to change the statement to false and break out of the do loop. A break statement can also cause a do loop to exit the do statement and continue with the next statement after the curly braces.

c) *For Loop*

A “for (expr1, epr2, expr3)”- sequence is equivalent to:

```
expr1;
while( expr2) {
        Command block
        expr3;
        }
```

A “for”-sequence is often used as a counter sequence.

```
for (i = 0; i < n; i++)
        . . .
```

Example:

```
#include “asuro.h”
int main(void) {
        int counter; // define a variable for counting
        for (counter =0;counter <10;counter ++) { // repeat ten
times:
        SerWrite(“Go ahead!\n”,10); // “Go ahead” message
        }
        MotorDir(FWD,FWD); // Both engines forward
        MotorSpeed(120,120); // Both engines running at around
half speed
        while (1) { // No more action!
        }
}
```

“while(1)” is equivalent to “for(;;)”, and both are eternal loops as the abort condition (in this case a
value 0) will never be reached.
Another loop sequence is a “do”-loop

```
do
command block
while( condition);
```

In contrast to a “while”-loop the condition will be checked after the command block. This programming sequence forces the command block to be executed at least one time.

6) Switch (case comparisons)

You can use the switch statement to compare one expression with others, and then execute a series of sub-statements based on the result of the comparisons. Here is the general form of a switch statement:

```
switch (test)
{
case compare-1:
        if-equal-statement-1
case compare-2:
        if-equal-statement-2
...
default:
        default-statement
}
```

The switch statement compares test to each of the compare expressions, until it finds one that is equal to test. Then, the statements following the successful case are executed. All of the expressions compared must be of an integer type, and the compare-N expressions must be of a constant integer type (e.g., a literal integer or an expression built of literal integers).

Optionally, you can specify a default case. If test doesn't match any of the specific cases listed prior to the default case, then the statements for the default case are executed. Traditionally, the default case is put after the specific cases, but that isn't required.

```
switch (x)
{
        case 0:
                puts ("x is 0");
                break;
        case 1:
                puts ("x is 1");
                break;
        default:
                puts ("x is something else");
                break;
}
```

Notice the usage of the break statement in each of the cases. This is because, once a matching case is found, not only are its statements executed, but so are the statements for all following cases. So the break statement is used to get out of the switch sequence.

Here is an example where the case comparison, commands to execute, and break statement are all on one line:

```
switch (cmd) {
        case RWD_KEY : PCBack(); SerWrite("back\n\r",7); break;
        case FWD_KEY : PCFront(); SerWrite("front\n\r",8); break;
        case LEFT_KEY : PCLeft(); SerWrite("left\n\r",7); break;
        case RIGHT_KEY : PCRight(); SerWrite("right\n\r",8); break;
        case STOP_KEY : PCStop(); SerWrite("stop\n\r",7); break;
}
```

You can also specify a range of consecutive integer values in a single case label, like this:
case low ... high:

This has the same effect as the corresponding number of individual case labels, one for each integer value from low to high, inclusive.

```
case 'A' ... 'Z':
case 1 ... 5:
```

Note: Be careful to include spaces around the ... otherwise it will not be interpreted properly.

7) Functions

Sometimes we need to use the same programming sequences at different locations in our programs. Of course we might repeat the writing or use a copy/paste-method for this purpose or we just define a function.

Sometimes we need to pass a few variables to a function, eg. telling our GoAhead () –function to use a given speed exactly, a certain lapse of time or a defined distance. We will use parameters for these details.

A definition block for a function always looks like:
Functions type Functions Name (ParameterType1 ParameterName1, ParameterType2 ParameterName2, ...)

Sometimes a function will even return a value. A good example is the HowManySwitchesHaveBeenActivated () -Function, returning a value, which will be defined somehow and somewhere within the function-block. The value is returned by a return-command at the end of the function-body. That's why functions end with return; or return number;.

A special function is the main () -function, defining the main body of a program. In ASURO the main () -function will be executed at switching ON. Of course every program must be supplied with a main () -function.

Now after some theory on data types and functions we would like to build a simple function, multiplying two 8-bit numbers and returning the result.

```
int Mult (char a, char b)
/* Function returns an int-value, is called Mult, and uses two char values
for input */
```

```
{ // Begin function
        int c; // Declaring variable c as an int
        c = a * b; // calculate c
        return c; // returning integer c
} // End of the function “Mult”
```

Now an example program, which will use the Mult-function we have defined before:

```
int main (void) // Function main always returns an int and will not
// input any parameter
{ // Begin Function „main“
        char mult1,mult2; // Defining two „char“-variables
        int result; // Defining an int-variable for the result of multiplying
        // variables mult1 und mult2
        mult1 = 2; // assignment
        mult2 = 10; // assignment
        result = Mult(mult1,mult2); // calling the previously defined
function “Mult”
return 0;
}
```

8) Number Systems Used in C

Let’s discuss a few details related to number systems, e.g. the decimal, the binary and the hexadecimal number system. As an introduction we will start with general details, valid for all number systems:

A number will be displayed as a sequence of digits, validated according to their sequence’s position. A base number, identical to the number of individual digit symbols, characterizes each number system.

a) Decimal Numbers

The decimal system with its base number 10 provides the digits 0..9. An example for an alternative coding of the number 1250 is:

$1205_{dec} = 1 \_ 10^3 + 2 \_ 10^2 + 0 \_ 10^1 + 5 \_ 10^0$

b) Hexadecimal Numbers

The example explains the basic idea: digits and the digits value in the decimal number system define a number. Here is another example:

The hexadecimal system uses 16 as base and of course it will provide 16 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The hexadecimal 12AC (to be interpreted as "1", "2", "A", "C") may be displayed as follows:

$12AC_{hex} = 1 \_ 16^3 + 2 \_ 16^2 + 10 \_ 16^1 + 12 \_ 16^0 = 4780_{dec}$

In C these Hex-numbers will be marked by a prefix `0x`, eg. `0x12AC`. Binary numbers will be coded in a similar way. The base number is 2, and we need just two digits 0 and 1. A conversion may be programmed by the formula:
$110110_{bin} = 1 \_ 2^5 + 1 \_ 2^4 + 0 \_ 2^3 + 1 \_ 2^2 + 1 \_ 2^1 + 0 \_ 2^0 = 54_{dec}$

c) *Binary Numbers*

In C binary numbers are indicated by a prefix `0b`, e.g. `0b110110`.

Digital systems usually use two states (on, off). These basic elements are called bits. Four bits make up a nibble and two nibbles a byte. At this point number systems get involved. A bitwise representation is simple enough in a binary number system, displaying just the states 0 and 1. One byte however already requires 8 individual binary digits and writing a lot of bytes is quite tedious. To make things easier programmers prefer to use the hexadecimal system. But why?

Let's look at the nibble. The maximum number, writeable in a nibble is:
$1111_{bin} = 1 \_ 2^3 + 1 \_ 2^2 + 1 \_ 2^2 + 1 \_ 2^0 = 15_{dec} = F_{hex}$

The nibble's complete number range may be displayed by one hexadecimal digit. This idea will vastly reduce our writing effort. Experienced programmers may quickly read the bit patterns inside the hexadecimal digits. A word (`int`) needs four hexadecimal digits.

The values for an `unsigned char` variable ranges from `0x00` up to `0xFF`, an `unsigned int` variable from `0x0000` to `0xFFFF`[2]. The following table displays a few decimal, hexadecimal, and binary numbers:

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100

13	D	1101
14	E	1110
15	F	1111
16	10	10000

9) Real Number Types

There are three data types that represent fractional numbers. While the sizes and ranges of these types are consistent across most computer systems in use today, historically the sizes of these types varied from system to system. As such, the minimum and maximum values are stored in macro definitions in the library header file float.h. In this section, we include the names of the macro definitions in place of their possible values; check your system's float.h for specific numbers.

a) Float

The float data type is the smallest of the three floating point types, if they differ in size at all. Its minimum value is stored in the FLT_MIN, and should be no greater than $1e^{-37}$. Its maximum value is stored in FLT_MAX, and should be no less than $1e^{37}$.

b) Double

The double data type is at least as large as the float type, and it may be larger. Its minimum value is stored in DBL_MIN, and its maximum value is stored in DBL_MAX.

c) Long double

The long double data type is at least as large as the float type, and it may be larger. Its minimum value is stored in LDBL_MIN, and its maximum value is stored in LDBL_MAX.

All floating point data types are signed; trying to use unsigned float, for example, will cause a compile-time error.

Here are some examples of declaring and defining real number variables:

```
float foo;
double bar = 114.3943;
```

The first line declares a float named foo but does not define its value; it is left uninitialized, and its value should not be assumed to be anything in particular.

The real number types provided in C are of finite precision, and accordingly, not all real numbers can be represented exactly. Most computer systems that GCC compiles for use a binary representation for real numbers, which is unable to precisely represent numbers such as, for example, 4.2. For this reason, we recommend that you consider not comparing real numbers for exact equality with the == operator, but rather check that real numbers are within an acceptable tolerance.

10) Converting Datatypes

Sometimes we will have to convert a variable from one datatype to another datatype. This may occur if we use different datatypes in several calculations for a variable. In these situations the compiler will output a warning and may use any datatype ad lib, which often results in strange results.

A cast, may control data type conversion. The cast syntax is simple and we merely need to insert the needed data type in round brackets before the variable name. The following example illustrates the syntax:

In our example we need to convert a variable `i` (originally declared as an `int`, but now to be converted to a *float* datatype, in order to do some accurate floating point calculations and then storing the result as an integer variable res, truncating it).

```
res= (int) ( ((float)i + 0.5) * 2.1 );
```

Using i=10 will result in res containing 22.

11) Type Casts

You can use a type cast to explicitly cause an expression to be of a specified data type. A type cast consists of a type specifier enclosed in parentheses, followed by an expression. To ensure proper casting, you should also enclose the expression that follows the type specifier in parentheses. Here is an example:

```
float x;
int y = 7;
int z = 3;
x = (float) (y / z);
```

In that example, since y and z are both integers, integer division is performed, and even though x is a floating-point variable, it receives the value 2. Explicitly casting the result of the division to float does no good, because the computed value of y/z is already 2.

To fix this problem, you need to convert one of the operands to a floating-point type before the division takes place:

```
float x;
int y = 7;
int z = 3;
x = (y / (float)z);
```

Here, a floating-point value close to 2.333... is assigned to x.

Type casting only works with scalar types (that is, integer, floating-point or pointer types). Therefore, this is not allowed:

```
struct fooTag { /* members ... */ };
struct fooTag foo;
unsigned char byteArray[8];

foo = (struct fooType) byteArray; /* Fail! */
```

12) Pointers and Vectors

Pointers belong to the most powerful C language standard elements. Misused, however, they support the creation of completely cryptic code.

A pointer is a variable that is describing the data's memory address instead of the data cell's contents. This is comparable with a box with a label "This way to the butter" and inside the box a note saying "Refrigerator, third level to the right". We will start with an example, declaring a pointer variable:

```
int *pointer;
```

In this example *pointer* is not a variable of type int. Instead the `*`-operator indicates it may contain an address for a variable of the int-type.

To fill a pointer, we use the address operator `&` which may only be applied to objects already available in the memory, and not to expressions, constants or register variables. The contents operator `*` is used to access the content of address variables, e.g. to display the address.

Here is an example:

```
char a = 1, b = 2;
char *cp   // cp is a pointer for a "char"
cp = &a    // cp now contains the address for variable a
b = *cp    // the address contents of cp are assigned to b
           // => b is set to 1
*cp = 0   // the address contents for cp will be set to 0
          // => a is set to 0
```

If we need to track data from the line tracing sensors or odometric sensors we will need vectors.
Their declaration is rather simple:

```
int lineData[2];
int odometrieData[2];
```

As you will recognize we will create two vectors (lData, oData) with 2 elements for linetracing and for odometrics. By calling ASURO Function (LineData(), OdometrieData ()) element [0] receives the value of the left sensor and element [1] receives the value of the right sensor.
We will demonstrate this method in an example:

If the right sensor receives more light than the left sensor, command1 should be executed otherwise command2.

```
int lData[2]; // Provide memory space for measurement results
LineData(lData); // Reading measurement data
if (lData[1] > lData[0])
          command1;
else
          command2;
```

To use the serial interface functions (SerWrite (), SerRead ()) we need character strings. These will be declared as follows:

```
char message[] = “This is a text string”
```

In order to send a text string in ASURO, a function SerWrite() is called with appropriate parameters. The first parameter contains the text string or the variable containing the text string, the second variable describes the number of characters to be sent, eg.:

```
SerWrite(message,20);
```

respectively

```
SerWrite(“This is a text“,14);
```

will transmit a message “This is a text” over the IR-Interface.

To receive characters, ARX ASURO uses a function SerRead (). The first parameter contains a variable in which the received characters are to be stored. The second parameter defines how many characters will be received and the third parameter defines a timeout: if within a certain time period (a number of processor clock cycles) no data has been received the SerRead () -function will be aborted. By using a number “0” however, the function will wait until all characters have been received. An example will illustrate this function.

ARX ASURO is to receive a message “Hi, here I am” by IR-interface. By a string definition

```
          char message [] = ”01234567890123456789”
```

We first allocate some storage space for the expected text. Of course the storage space must be big enough to contain the expected message.

```
SerRead (message,13,0)
```

Read 13 characters and wait until 13 characters have been received. We now consider the text string “Hi, here I am” has been sent. The function will now overwrite the first 13 characters of the predefined string message with “Hi, here I am”, resulting in a string:

Hi, here I am 3456789

13) Addressing Ports

When addressing parts of the microcontroller, we use the port numbers. This table shows the ports and their corresponding pin locations on the microcontroller chip.

Port	Pin-Nr.	I/O Function	Notes
PB0	14	Output Status-LED green	
PB1	15	OC1A PWM for left Motor	
PB2	16	OC1B PWM for right Motor	
PB3	17	OC2 36kHz-Modulation for IR communication	
PB4	18	Output forward/backward control for left motor	
PB5	19	Output forward/backward control for right motor	
PB6	9	XTAL1 8MHz Resonator	The internal oscillator is too inaccurate for IR-communication.
PB7	7	XTAL2 8MHz Resonator	
PC0	23	ADC0 / Output Odometrics left / left Back-LED	No coincident Odometrics and left Back- LED allowed
PC1	24	ADC1 / Output Odometrics rigth/ right Back-LED	No coincident Odometrics and right Back- LED allowed.
PC2	25	ADC2 Phototransistor bottomside left	Freely available (Input/Output/ADC2) extension board.
PC3	26	ADC3 Phototransistor bottomside rechts	Freely available (Input/Output/ADC3) extension board.
PC4	27	ADC4 Used for reading the Sensorswitches	NA
PC5	28	ADC5 Supply voltage monitoring	NA
PC6	1	Reset none / set at supply voltage level	will be needed for reprogramming the bootloader in the processor.
PD0	2	RXD	Receive serial data
PD1	3	TXD	Send serial data
PD2	4	Output Status-LED red	Freely available (Input/Output/INT0) extension board.
PD3	5	INT1 Interrupt 1 for tactile sensors	For interrupt controlled reading tactile sensors.
PD4	6	Output	Forward / backward control right motor
PD5	11	Output	Forward / backward control left
PD6	12	Output Front-LED	Freely available (Input/Output/AIN0) extension board.

Port	Pin-Nr.	I/O Function	Notes
PD7	13	Output / Input Toggle Odometrics / Back-LEDs	No coincident Odometrics and left Back-LED allowed.

14) Bit Manipulations

Our microcontroller is able to operate using only two states (current on and current off), and this kind of system needs some basic knowledge of Boolean Algebra in order to manipulate individual bits in the processor's register. To do so the C-language uses operators, which have to be applied to number operands (`char`, `short int`, `long int`; both `signed` and `unsigned`).
Additionally operators for logical combinations of Boolean variables will be used. We will have a closer look to these operators.

a) *AND-Operator*

C-Operator: `&`
Resulting in 1, if all conditions are 1.
Combining two bits A and B in a Boolean `&`-expression results in:

A	B	X
0	0	0
1	0	0
0	1	0
1	1	1

Example:
17 & 13 = 1

b) *OR-Operator*

C-Operator: `|`
Resulting in 1, if at least one conditions is 1.
Combining two bits A and B in a Boolean `|`-expression results in:

A	B	X
0	0	0
1	0	1
0	1	1
1	1	1

Example:
17 | 13 = 29

c) *Exclusive-OR-Operator*

C-Operator: ^
Resulting in 1, if exactly one conditions is 1.

Combining two bits A and B in a Boolean ^-expression results in:

A	B	X
0	0	0
1	0	1
0	1	1
1	1	0

Example:
17 ^ 13 = 30

d) Bit-Complement

C-Operator: ~
Resulting in 1, if the conditions is 0.
The operator inverts bits, eg. a "0" will be converted to "1" and vice versa.
Example:
$00101011_{bin} = 11010100_{bin}$
~0x2B = 0xD4

e) Bitshift to the left

C-Operator: <<
A bitshift to the left by n bits will shift the bits by the specified number n of bits. Incoming new bits from the right will be "0". An example demonstrates the operator in a binary and hexsystem:

$1011_{bin} << 3 = 1011000_{bin}$
0x0B < < 3 = 0x58
A bitshift to the left by n bits is equivalent to a multiplication by 2_{n}.

f) Bitshift to the right

C-Operator >>
A bitshift to the right by n bits will repeat moving the number by the specified number n of bits to the right. Additional new bits will be filled with "0"[3]. An example demonstrates the operator in a binary and hex-system:

$10110001_{bin} >> 4 = 00001011_{bin}$
0xB1 > > 4 = 0x0B
A bitshift to the right by n bits is equivalent to a division by 2_{n}.

g) Application of Bit Manipulation

How are we going to use these operators? The next section will describe the individual registers (8-bit wide memory cells), which are relevant for the ARX ASURO. We will learn it is important to know how to set or reset individual bits in a register, and how to apply the described operators. Whenever we need to set an individual bit, this may be done by a simple command.

For example, in order to set bit 5 in a register called `DDRB`:

```
DDRB = DDRB | (1<<5)  or
DDRB = DDRB | 0x20
```

In C we may also write:

```
DDRB |= 0x20
```

Setting bit 3 and bit 7 simultaneously requires:

```
DDRB |= (1< <3) | (1< <7)
```

It is a little bit more complicated to reset a bit. As an example we will reset bit 5:

```
DDRB = DDRB & ~(1<<5)
```

or:

```
DDRB &= ~(1< <5)
```

Resetting bit 3 and 7 requires:

```
DDRB &= ~((1<<3) | (1<<7))
```

or also possible:

```
DDRB &= ~0x88
```

15) Conditional Operators

Additionally to manipulate individual bits by Boolean operators we may also apply conditional operators to Boolean variables. Unfortunately, C uses different coding for describing bit manipulation operators and for logical conditional operators to Boolean variables. It's funny, but it's true:

a) AND-Operator

C-Operator: `&&`
Result is true, if all parameters are true.
Combining two parameters A and B by `&&` will result in:

A	**B**	**X**
false	*false*	*false*
true	*false*	*false*
false	*true*	*false*
true	*true*	*true*

Example:
if ((a>-5)&&(a<5)) {...}
// If a is greater than -5 and less than 5...

b) OR-Operator

C-Operator: `||`
Result is true, if at least one parameter is true.
Combining two conditional operators A and B by `||` will result in:

A	**B**	**X**

false	*false*	*false*
true	*false*	*true*
false	*true*	*true*
true	*true*	*true*

Example:
while ((v_running)||(counter<20)) {...}
// continue as long as counter<20 or variable v_running is
// active

c) Bit-Complement

C-Operator: !
The result is true, if the condition is false
Example:
while (!Keypressed()) {...} // As long as no key is pressed

16) Registers

This is an advanced topic for experienced programmers.

Register I/O-Ports:

Data Direction Register

Bit	7	6	5	4	3	2	1	0
Pin	DDRx7	DDRx6	DDRx5	DDRx4	DDRx3	DDRx2	DDRx1	DDRx0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Port Data Register

Bit	7	6	5	4	3	2	1	0
Pin	PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Port Input Pin

Bit	7	6	5	4	3	2	1	0
Pin	PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

This section is devoted to the processor's most important registers, but will be restricted to objects relevant to ARX ASURO. Single ports - apart from their special functions - may be configured as input resp. output terminal, inputting or outputting voltage levels. Each port provides three registers (see register I/O Ports table).

The data-direction-register DDRx (x=B,C,D) annotates which pins in a port are to be output or input terminals. A logical '1' at the individual bit-location will set the corresponding pin to be an output terminal and a logical '0' at will set the corresponding pin to be an input terminal.

In order to program an output pin to electrically high (resp. low), we need to set the corresponding bit in the port-data-register *PORTx* to '1' (resp. '0').

Reading the levels at input pins we need to read the corresponding bits in the register PINx. But don't make the mistake, reading the *PORTx*-register again, which will result in reading the most recent information written into the *PORTx*-register and not the applied logical level to the pin.

If a pin is programmed to be an input, setting a bit in PORTx to '1' will result in activating an internal pull-up resistor.

This looks rather complicated, but an example will make it transparent. We want to switch on the green Status-LED. A glance at the schematic diagram reveals the "green" terminal is found at PB0, and that the following steps will be required:

Select pin 0 in Port B to become an output pin: DDRB |= (1< <PB0)
Set this pin to '1': PORTB |= (1< <PB0)
Switch off the LED, if required: PORTB &= ~(1<<PB0)

We can use a similar process to activate the other LEDs and motors. Talking about motor control, we will also learn how to control the motor speed. The corresponding terminals are PB1 resp. PB2 (see schematic diagram for ARX ASURO).

These will be programmed to be outputs (DDRB |= (1< <PB1) | (1< <PB2)). Additionally this I/O-terminal will also be used as OC1A resp. OC1B. In order to build a PWM (Pulse Width Modulation) - system using these pins we need to reset and set a few special pins in other registers (TCCR1A, TCCR1B).

A detailed description of these registers would be beyond the scope of this book. The relevant details however may be found in the ARX ASURO microcontroller specifications and the A/D Controller documentation.

Registers ADCSRA, ADMUX, ADCL and ADCH are controlling A/D-conversions, eg. starting stopping, programming conversion rates and sampling measurement values.
Experts may be interested in the registers for communication with the internal EEPROM and for using an additional timer circuit.

17) Interrupts

If we program the ARX ASURO at the register level, we will need to use an interrupt. But before doing so, we want to know what an interrupt really is. An interrupt, interrupts the processors workflow. The `main` program is stopped immediately. The processor now jumps into an ISR (Interrupt Service Routine) - which of course has to be programmed before - and having finished this procedure the processor will continue the `main` program as if nothing had been happening.

Atmel ATmega8 provides the following interrupts:

SIG_INTERRUPT0	// External Interrupt Request0
SIG_INTERRUPT1	// External Interrupt Request1
SIG_OUTPUT_COMPARE2	// Timer/Counter2 Compare Match
SIG_OVERFLOW2	// Timer/Counter2 Overflow
SIG_INPUT_CAPTURE1	// Timer/Counter1 Capture Event
SIG_OUTPUT_COMPARE1A	// Timer/Counter1 Compare Match A
SIG_OUTPUT_COMPARE1B	// Timer/Counter1 Compare Match B
SIG_OVERFLOW1	// Timer/Counter1 Overflow
SIG_OVERFLOW0	// Timer/Counter0 Overflow
SIG_SPI	// Serial Transfer Complete
SIG_UART_RECV	// USART, RX Complete
SIG_UART_DATA	// USART, Data Register Empty
SIG_UART_TRANS	// USART, TX Complete
SIG_ADC	// ADC Conversion Complete
SIG_EEPROM_READY	// EEPROM Ready
SIG_COMPARATOR	// Analog Comparator
SIG_2WIRE_SERIAL	// Two-wire Serial Interface
SIG_SPM_READY	// Store Program Memory Ready

Predefining interrupts and many port pins for special applications the ARX ASURO user's interest may be restricted to the following interrupt vectors:
SIG_INTERRUPT0: May be used by an optional additional extension board.
SIG_INTERRUPT1 : Will be used by the collision switches.
By using the ARX ASURO function StartSwitch() the users may activate resp. by using StopSwitch() they may deactivate the corresponding ISR ad lib.

SIG_OUTPUT_COMPARE2: already is activated for the IR-communication and is used in the Sleep()-Function.
SIG_UART_RECV, SIG_UART_DATA, SIG_UART_TRANS: You may use these interrupts in programming an interrupt-controlled infrared communication (for the infrared interface).
SIG_ADC: Available for interrupt-controlled reading an A/D-converter in some applications.
SIG_EEPROM_READY: Working with the EEPROM, the interrupt may efficiently control the waiting cycles for writing the EEPROM.

Before writing our first ISR (Interrupt Service Routine) we will need some standard information for these functions:

- The ISR of course should be short, to allow the main program main() to do as much as possible.
- Of course the ISR should be completed before the next interrupt arrives. The ARX ASURO Processor does not allow another ISR while processing a preceding ISR, if the function is defined as a SIGNAL.
- Please don't overload your program with interrupt functions, which may result in funny effects. ISR may collide and the individual ISR flow may be confusing.
- Interrupts are prioritized individually, resulting in a preferred interrupt if two occur at the same moment.
- The interrupt list above is sorted from highest to lowest priority. Variables to be changed inside an ISR and accessible for external use must be set to `volatile` (disabling the compilers optimizer for register variables).
- Normally interrupt routines use global variables (attributed as volatile), not returning variables or reading parameters.

Let's analyze the following example:
We will find the function Sleep() among the asuro.h functions library.

```
volatile unsigned char count72kHz;
. . .
void Sleep(unsigned char time72kHz)
{
count72kHz = 0;
while (count72kHz < time72kHz);
}
```

The function is inactive as long as count72kHz is less than `time72kHz`, but where do we alter count72kHz? This will be done by an interrupt routine, triggered by a timer, even when the program is stuck in a while-loop. Normally in our processor we activate an interrupt by a keyword SIGNAL , specifying the corresponding interrupt vector in round brackets. And somewhere our source code will contain an ISR like:

```
SIGNAL (SIG_OUTPUT_COMPARE2)
{
count72kHz ++;
}
```

SIG_OUTPUT_COMPARE2 demonstrates the use of Timer/Counter2, triggering an interrupt each time the timer reaches an overflow state. This timer controls the 36kHz square wave modulation signal and will overflow twice in each 36kHz signal period, resulting in generating an interrupt each
1
36kHz_2 = 13:88889_s and simultaneously incrementing `count72kHz` by one. Of course the register controlling Timer/Counter2 has to be programmed to generate a 36kHz signal (although needed for ASUROs IR-Transmitter 2.4) and to activate the appropriate interrupt.

There is one important detail to be considered: to avoid problems we sometimes define program flows, which are not to be interrupted at all. To control interrupting we need two functions to globally activate or deactivate interrupts. At program start interrupts generally are deactivated and a program command will have to activate interrupting.

Activate interrupts:

```
sei();
```

Deactivate interrupts

```
cli();
```

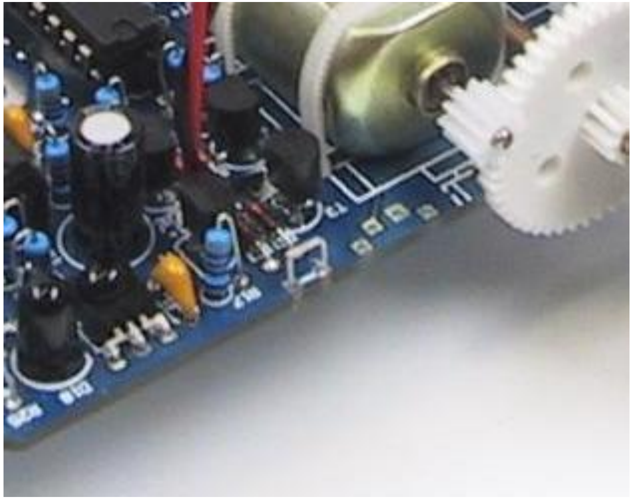
Reactivating interrupts without knowing if they have been activated before disabling them is very bad style and may result in problems. Better is to memorize the processor's status register, disable the interrupts and to reactivate them write back the memorized status register. At the beginning of the program the interrupts may be activated with sei().

C. Modifications to the Robot

Here are few ideas for minor modifications which may be made to the ARX ASURO robot. *Note these modifications are not supported as part of this material. You are on your own with making these changes. Look online for support groups which may assist in answering questions.*

1) Ground clip

We often need to check voltages with an oscilloscope or multimeter at the top or bottom of the system. A simple ground clip providing the GND-signal connection is helpful. Of course ARX ASURO provides the contact - if you solder a tiny clip to the appropriate contact on the printed circuit board - directly along the resistor R17.



2) Modification for image recognition

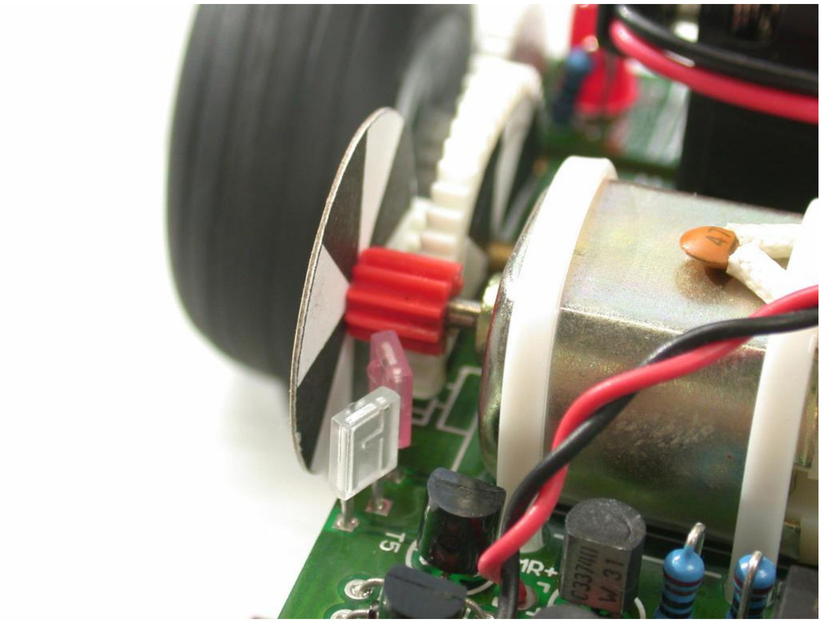
Experiments involving several ARX ASUROs may require observation of the scene by a camera mounted at the ceiling. Identifying individual vehicles may be accomplished by using labels or different colors.
An alternative method applies the line tracing LED (D11) mounted on top. Then we get – together with the back-LEDs - an isosceles triangle that can be used to track ARX ASUROs position and heading with methods of image recognition (which again would exceed the scope of this book).

3) Faster odometer

For some applications we need a higher resolution for wheel speed measurements. Normally we register 8 impulses for each rotation of the first gear cogwheel corresponding to 40 impulses for a tire wheel rotation. You may however modify the reflection decoder in order to register the pinion rotations at the motor axle, resulting in 200 impulses for each tire rotation.

The modification implies a removal of the LEDs and phototransistors (T11, T12, D13, D14) in the reflection decoder and mounting these (or alternatively new components) somewhat higher in the holes along the pinions (see Fig. 10.2 left), taking care to locate the tiny spot at the pinion center level. Now we prepare two new paper or cardboard copies for the decoder discs, drilling 2 mm holes into the center for each of them, and gluing these to pinion.

Basically decoding the wheel speed will be using the same procedure, but we will be receiving five times more impulses for each wheel rotation. The modification is only worth the trouble if high resolution is really required. A disadvantage is in having to remove the decoding disc(s) when removing the gears.



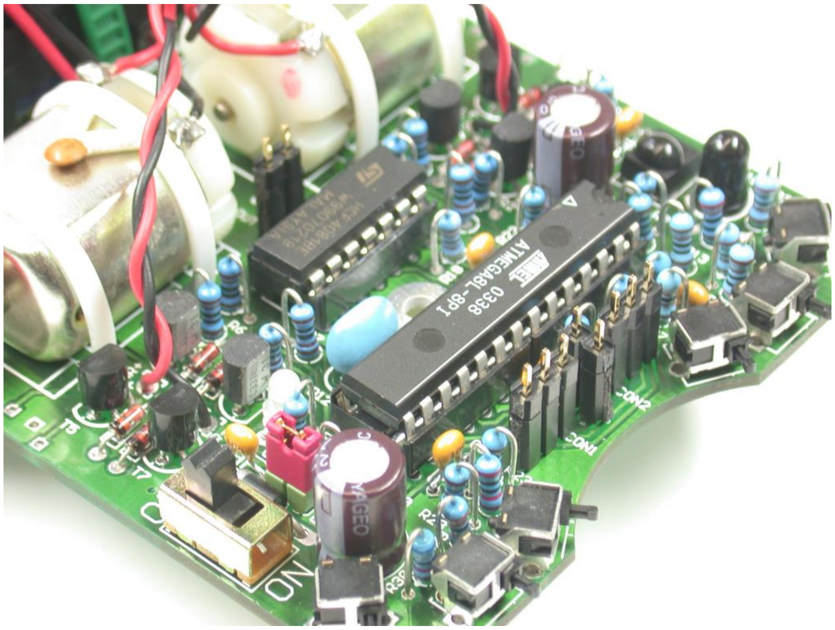
D. Modifying ARX ASURO's main PCB

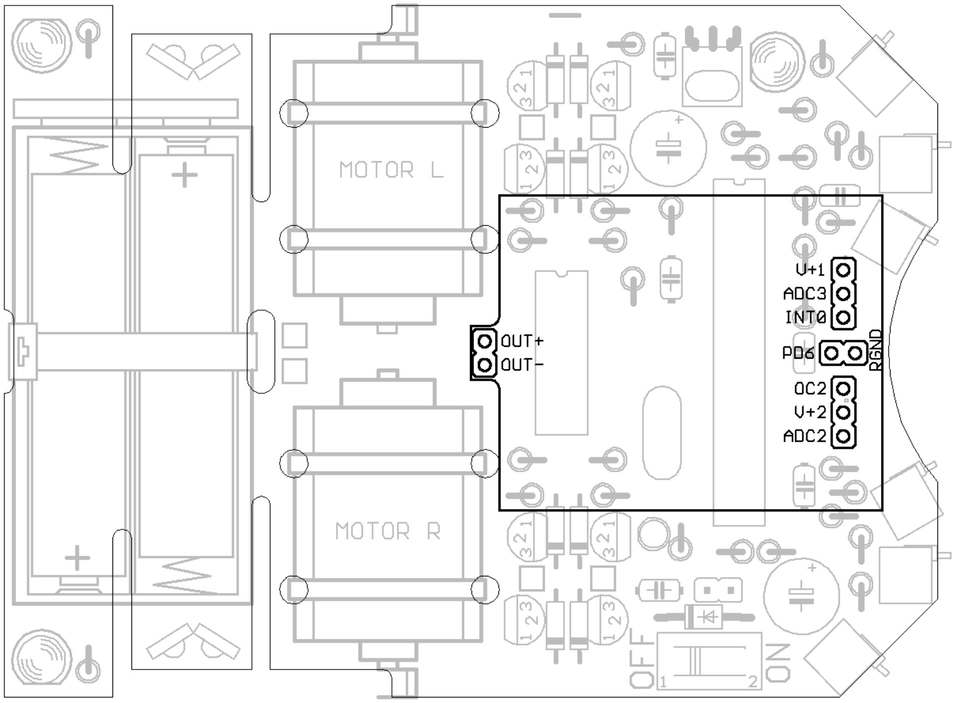
Unfortunately we need some mechanical modification for additions. First of all we remove the pingpong ball carefully, as we will put it back again after modifications. Adding an extension board also requires a removal of the line tracing sensors and the LED. The soldering holes need to be freed in order to insert the contact strips.

Please check the location again for all parts under the extension board. They should be mounted at a low level, to prevent mechanical collision and electrical short circuits.

Now split a small contact strip, by carefully using a sharp edge or pincers into two dual pin plugs, and two threefold pin plugs. Use the same procedure for the contact strips.

Plug the pin plugs into the corresponding contact strips. Insert the three-pinned contact strips - including the threefold pin plugs, but without applying solder yet - at the component side of the main PCB into the holes of the phototransistors. The contact strips must be located at the ARX ASURO main board. Now you understand why we need an additional hole.

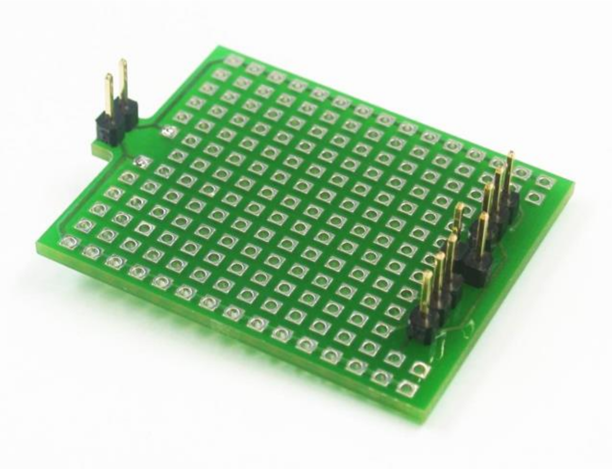




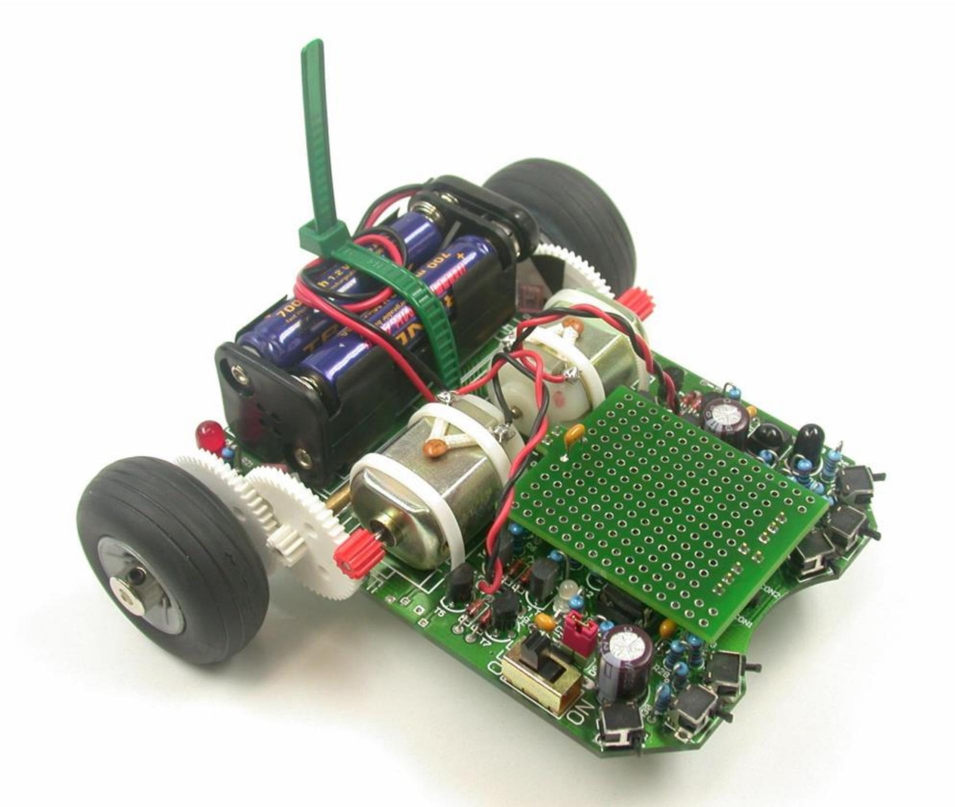
One dual contact strip fits into the LED holes at the component side and the other one into the holes labeled OUT+ and OUT- (see diagram above). Again we take care not to be soldering yet and to locate the contact strips at the ARX ASURO main board.

Now insert the extension board - locating the soldering/wiring side at the bottom - at the pin contact strips. After inserting successfully, solder the extension board beginning at the extension board and subsequently at the ARX ASURO main board. Wait a few seconds to allow the solder to cool down to normal temperature and the extension board may easily be unplugged from, or re-plugged onto, the main board.

As a finishing touch we insert a supplied 100nF capacitor into the holes along the supply pins (OUT+ und OUT-). Take care to place the capacitor at the topside of the board (see green board below)!



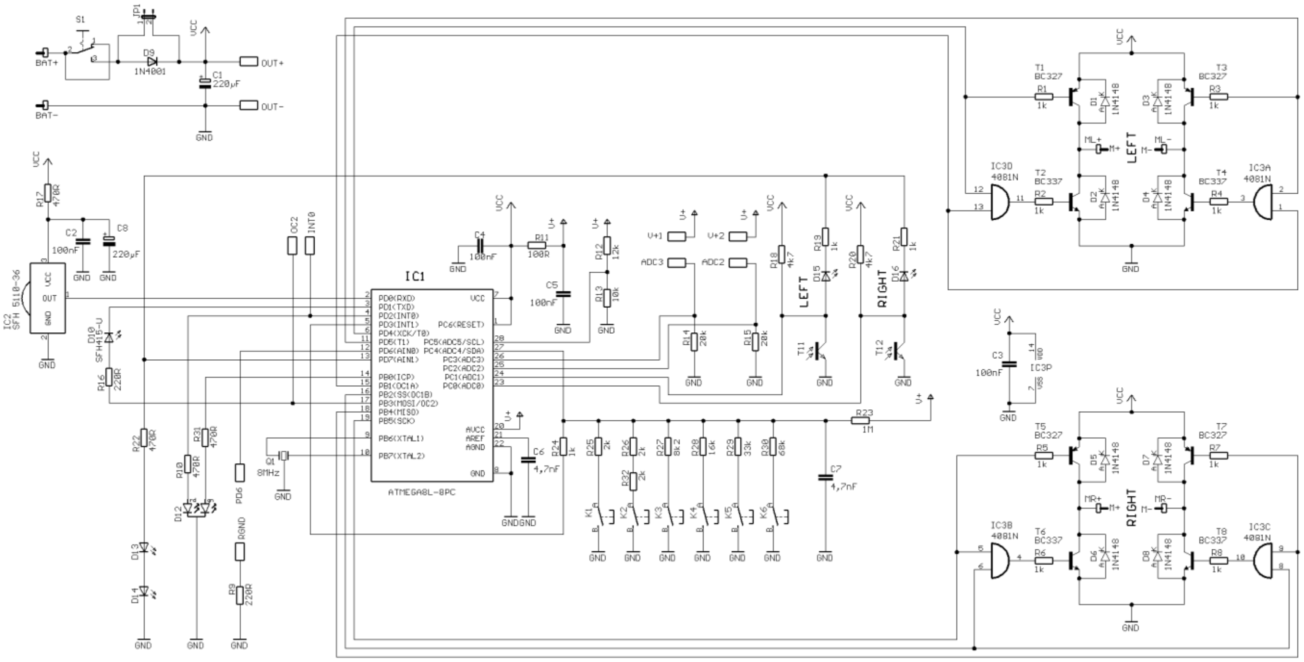
Extension board with pin contacts and capacitor

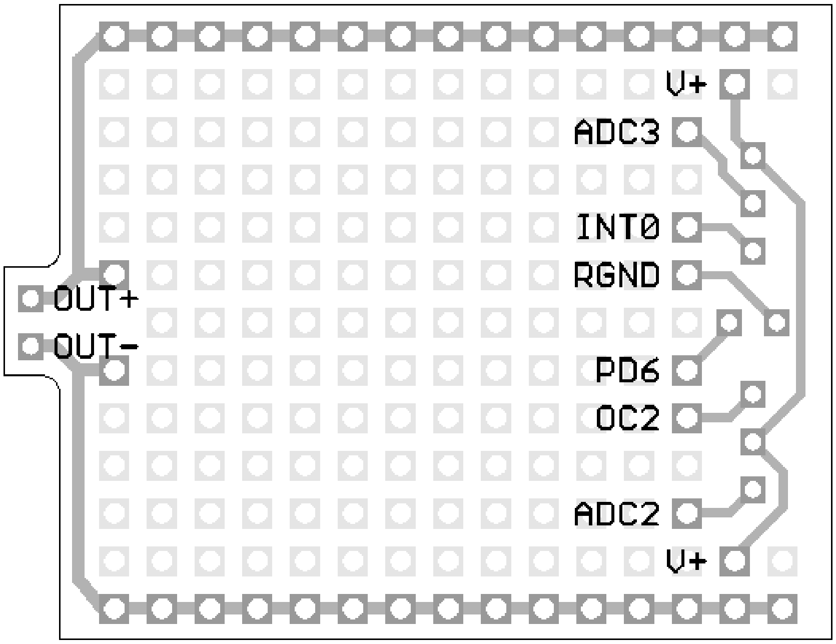


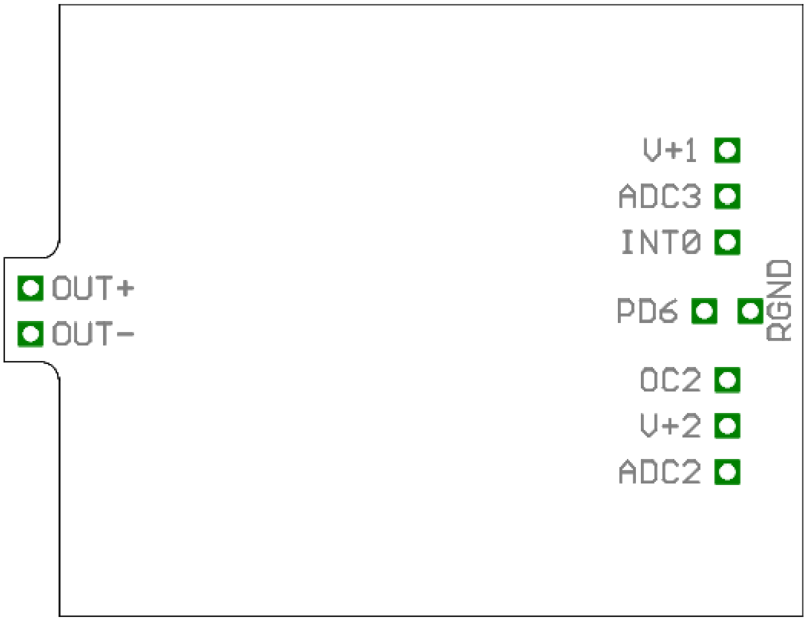
Extension board add-on to ARX *ASURO with a capacitor*

1) Options for the extension board

Which options are available by adding the extension board?
A number of important processor pins are freely available to the extension unit. See the modified ARX ASURO schematic including the additional pin contact signals and sketches for the pin signals for the extension board. The extension board provides a contact to the nearest holes, whereas the signals OUT+ and OUT- each are contacting a complete row.







A brief description of the signal pins:

OUT+
The signal provides a positive supply voltage for the extension board, supplying - according to the battery voltage level - between +4.0V and +5.5V.
OUT This signal provides the supply ground level for extension board: 0V resp. GND
V+1, V+2

These pin terminals supply the analog supply voltages for the main board and may also be used as reference voltages for the A/D converter. At the ARX ASURO main board the supplies are decoupled from course requires a maximum drain current of a few Milliamps to prevent the voltage from collapsing.

If you need a filtered voltage at higher current levels, a separate filter at the extension board may be assembled (e.g. as an LC filter, feeding its output voltage into V+).
At the extension board V+1 and V+2 have been connected to V+ .

RGND
Will be connecting by R9 to the GND signal and should be connected to the GND signal at the extension board. By bridging resistor R9 with some solder at the main board the pin may be used as analog ground signal, e.g. as ground level for sensitive analog circuits.

PD6(AIN0)
Pin PD6 replaces the former front-LED and may be programmed by processor commands to be an input, an output or a negative input terminal to the analog comparator.

INT0(PD2)
At the PCB terminal INT0 is interconnected to processor pin PD2, and may be used either as an input, an output or an interrupt input terminal. A series resistor also interconnects this pin to the status LED.

Programming this pin as an output pin for other purposes may result in strange effects if the output is flashing the absent LED, while booting the processor or in other programs.
If the pin is programmed as an input the driver module will have to supply enough current (10mA) to activate the parallel LED as well.

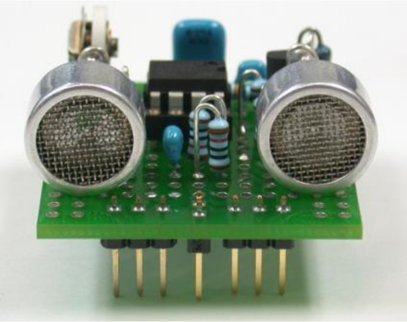
OC2(PD3)
OC2 is interconnected to a digital processor pin (PB3) as well and may be used as an input, an output, or a timer output terminal for timer2. Remember OC2 is also used for generating the 36kHz modulation signal in the infrared communication module. The system will generate a corresponding signal while flashing and while booting. If you use this pin the infrared communication channel is disabled. You should not use the pin as an input terminal, in order to allow consistent program access to the processor system by the infrared communication channel.

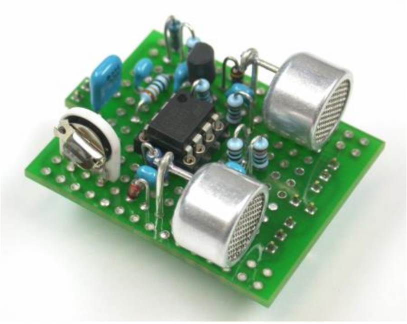
ADC3 (PC3)
ADC3 is interconnected to A/D converter pin 3 resp. port terminal PC3, and may be used an analog or digital input, or as a digital output. Please remember the built-in pulldown resistor of 20k. If you use the pin as an output, consider that the current will have to be supplied from the analog supply voltage V+.

ADC2 (PC2)
ADC2 (PC2) is interconnected to A/D converter pin 2 resp. portterminal PC2. The pin shares all other details with ADC3.

But why these strange dimensions?
Once upon a time these electronic parts used to be placed in inch-oriented grids and so we also apply a 1/80-inch grid in our design, which complicates dimension numbers in mm.

2) Example UltraSound Sensor Add-on





Example Extension Board with Ultrasound Sensors

Ultrasonic distance measurement systems use the simple basic echo principle from the ASDIC (Allied Submarine Detection Investigation Committee) system, transmitting a short, high-frequency beep. Not unlike an RF radar signal this sound signal will travel in all directions at the speed of sound, and will be reflected by objects in its path. The echos return to the transmitter and may be detected.

Bill of material for Ultrasound-Sensor

1 x strip tenfold or more (to be splitted)
1 x contact strip tenfold or more (to be splitted)
6 x 100nF ceramic
4 x 10k Ω 1/4W 5%
1 x 100 Ω 1/4W 5%
1 x 1k Ω 1/4W 5%
1 x 100k Ω 1/4W 5%
1 x 20k Ω 1/4W 5%
1 x 470k Ω 1/4W 5%
1 x Trimmer 1M
1 x Ultrasonic-transmitter module e.g.: Polaroid 40LT12
1 x Ultrasonic-receiver module e.g. : Polaroid 40LR12
1 x BC547 B or C
1 x 1N4148
1 x TS912IN or similar type

(search for additional information and program examples online for this project)

E. Tables

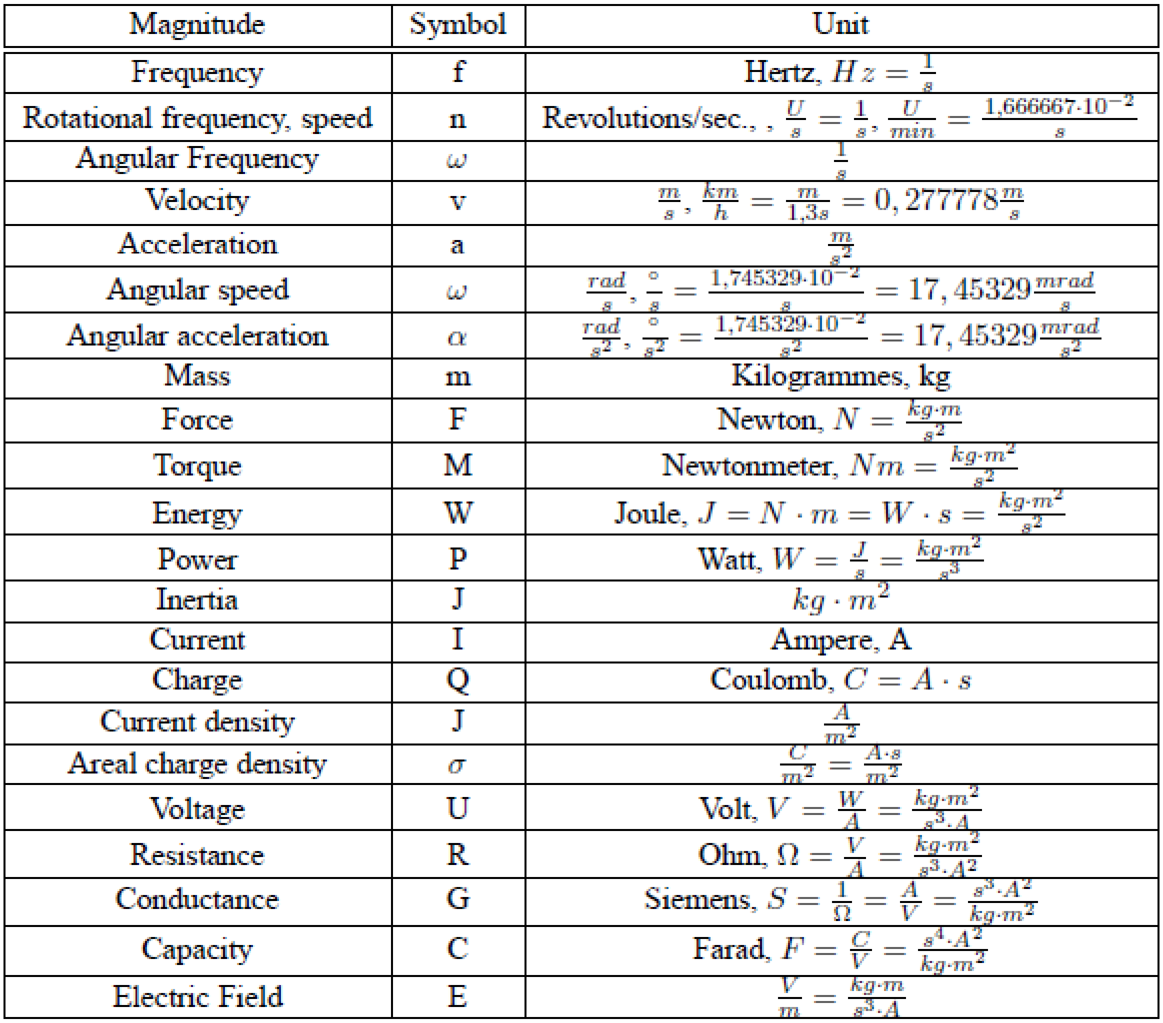
1) ASCII Table

Hex	Dec	Character	Hex	Dec	Character	Hex	Dec	Character	Hex	Dec	Character
00	0	NUL	20	32	SP	40	64	@	60	96	‘
01	1	SOH^A	21	33	!	41	65	A	61	97	a
02	2	STX^B	22	34	”	42	66	B	62	98	b
03	3	ETX^C	23	35	#	43	67	C	63	99	c
04	4	EOT^D	24	36	$	44	68	D	64	100	d
05	5	ENQ^E	25	37	%	45	69	E	65	101	e
06	6	ACK^F	26	38	&	46	70	F	66	102	f
07	7	BEL^G	27	39	’	47	71	G	67	103	g
08	8	BS^H	28	40	(	48	72	H	68	104	h
09	9	TAB^I	29	41	)	49	73	I	69	105	i
0A	10	LF^J	2A	42	*	4A	74	J	6A	106	j
0B	11	VT^K	2B	43	+	4B	75	K	6B	107	k
0C	12	FF^L	2C	44	,	4C	76	L	6C	108	l
0D	13	CR^M	2D	45	-	4D	77	M	6D	109	m
0E	14	SO^N	2E	46	.	4E	78	N	6E	110	n
0F	15	S^O	2F	47	/	4F	79	O	6F	111	o
10	16	DLE^P	30	48	0	50	80	P	70	112	p
11	17	DC1^Q	31	49	1	51	81	Q	71	113	q
12	18	DC2^R	32	50	2	52	82	R	72	114	r
13	19	DC3^S	33	51	3	53	83	S	73	115	s
14	20	DC4^T	34	52	4	54	84	T	74	116	t
15	21	NAK^U	35	53	5	55	85	U	75	117	u
16	22	SYN^V	36	54	6	56	86	V	76	118	v
17	23	ETB^W	37	55	7	57	87	W	77	119	w
18	24	CAN^X	38	56	8	58	88	X	78	120	x
19	25	EM^Y	39	57	9	59	89	Y	79	121	y
1A	26	SUB^Z	3A	58	:	5A	90	Z	7A	122	z
1B	27	Esc	3B	59	;	5B	91	[	7B	123	{
1C	28	FS	3C	60	<	5C	92	\	7C	124	|
1D	29	GS	3D	61	=	5D	93	]	7D	125	}
1E	30	RS	3E	62	>	5E	94	^	7E	126	~
1F	31	US	3F	63	?	5F	95	_	7F	127	DEL

2) Common Units

Tera	T	10^{12}
Giga	G	10^{9}
Mega	M	10^{6}
kilo	k	10^{3}
hundreds		10^{2}
tens		10^{1}
milli	m	10^{-3}
micro		10^{-6}
nano	n	10^{-9}
piko	p	10-^{12}

3) Symbols



Magnitude	Symbol	Unit
Frequency	f	Hertz, $Hz = \frac{1}{s}$
Rotational frequency, speed	n	Revolutions/sec., , $\frac{U}{s} = \frac{1}{s}, \frac{U}{min} = \frac{1{,}666667 \cdot 10^{-2}}{s}$
Angular Frequency	ω	$\frac{1}{s}$
Velocity	v	$\frac{m}{s}, \frac{km}{h} = \frac{m}{1{,}3s} = 0,277778\frac{m}{s}$
Acceleration	a	$\frac{m}{s^2}$
Angular speed	ω	$\frac{rad}{s}, \frac{\circ}{s} = \frac{1{,}745329 \cdot 10^{-2}}{s} = 17,45329\frac{mrad}{s}$
Angular acceleration	α	$\frac{rad}{s^2}, \frac{\circ}{s^2} = \frac{1{,}745329 \cdot 10^{-2}}{s^2} = 17,45329\frac{mrad}{s^2}$
Mass	m	Kilogrammes, kg
Force	F	Newton, $N = \frac{kg \cdot m}{s^2}$
Torque	M	Newtonmeter, $Nm = \frac{kg \cdot m^2}{s^2}$
Energy	W	Joule, $J = N \cdot m = W \cdot s = \frac{kg \cdot m^2}{s^2}$
Power	P	Watt, $W = \frac{J}{s} = \frac{kg \cdot m^2}{s^3}$
Inertia	J	$kg \cdot m^2$
Current	I	Ampere, A
Charge	Q	Coulomb, $C = A \cdot s$
Current density	J	$\frac{A}{m^2}$
Areal charge density	σ	$\frac{C}{m^2} = \frac{A \cdot s}{m^2}$
Voltage	U	Volt, $V = \frac{W}{A} = \frac{kg \cdot m^2}{s^3 \cdot A}$
Resistance	R	Ohm, $\Omega = \frac{V}{A} = \frac{kg \cdot m^2}{s^3 \cdot A^2}$
Conductance	G	Siemens, $S = \frac{1}{\Omega} = \frac{A}{V} = \frac{s^3 \cdot A^2}{kg \cdot m^2}$
Capacity	C	Farad, $F = \frac{C}{V} = \frac{s^4 \cdot A^2}{kg \cdot m^2}$
Electric Field	E	$\frac{V}{m} = \frac{kg \cdot m}{s^3 \cdot A}$