SRS Robot Level 1 Kit

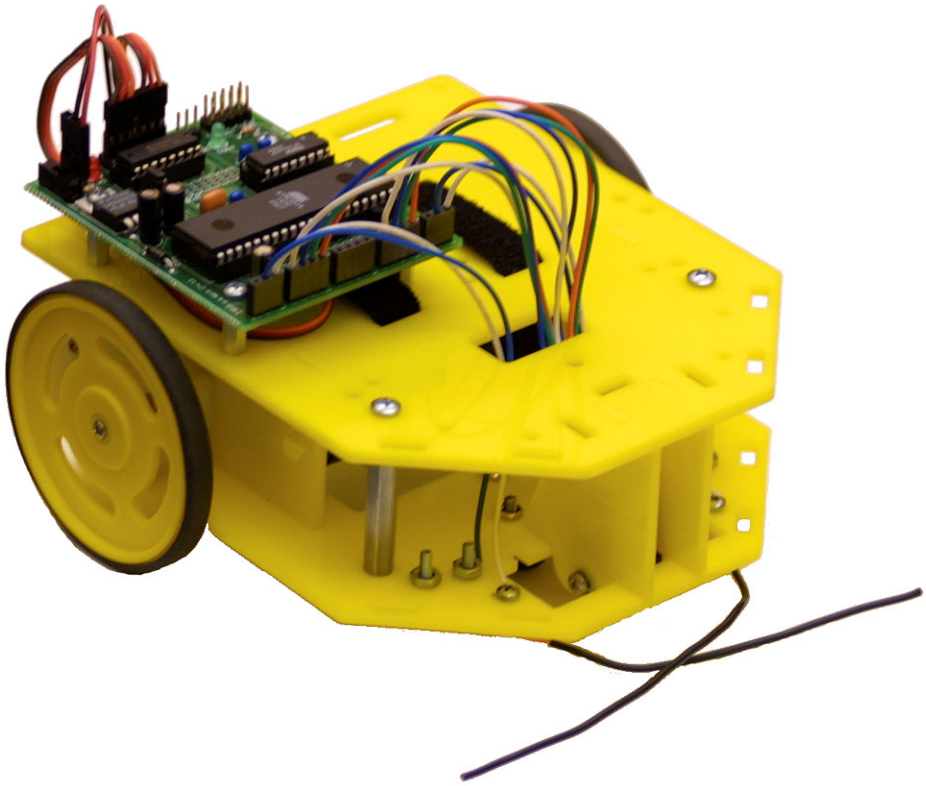


Table of Contents

Getting Started

Congratulations on your purchase of a Seattle Robotics Society Level 1 Robot Kit!

Here are suggested steps for getting your robot assembled and running.

Read through the “Computer Requirements” and “Recommended Equipment” sections to be sure you have the materials you need.

Compare your kit’s parts with the lists in the “Kit Contents” section to verify that you can correctly identify each of the parts in the kit.

The sections after that work well as a tutorial, with step-by-step instructions for assembling the controller board and chassis, guidance for setting up the programming software on your computer, explanations of the basics of programming the Atmel ATmega16 microcontroller, and an introduction the programs. The programs themselves contain comments to explain the concepts in more detail.

The sections at the end, “AVR Robot Controller 1.1 Hardware Description” and “AVR Robot Controller 1.1 Schematic,” are included for reference.

Computer Requirements

To program the robot, you will need *one* of the following connections:

- a DB-25 **parallel port** -- this will work with the programming cable included in the kit. (Note that it will most likely *not* work to use a USB-to-parallel converter; they do not provide full parallel port functionality, just what's necessary for printing!)
- a DB-9 **serial port** and an **AVR-ISP** programmer -- the AVR-ISP can be purchased from Digi-Key (part number ATAVRISP-ND) for $29.
- a **USB port** with a **USB-to-serial converter** and an **AVR-ISP** programmer -- USB-to-serial converters are available from a variety of places, including Fry's Electronics; the AVR-ISP can be purchased from Digi-Key (part number ATAVRISP-ND) for $29.

To communicate with the robot, you will need either:

- a DB-9 **serial port**, or
- a **USB port** with a **USB-to-serial converter**.

Recommended Equipment

The robot is powered with **6 AA batteries**, which are not included in the kit. We recommend rechargeable batteries (NiMH or NiCd).

The following items will be necessary (or useful) for construction. Please bring as many of these items as possible to the workshop sessions. There will be a small quantity available to use during the workshop.

- Safety glasses
- Soldering iron
- Solder (e.g. 60/40 0.032" dia rosin core solder, Radio Shack 64-005)
- Wire cutter/stripper
- Side cutters (for trimming leads on soldered components)
- Needlenose pliers
- Pliers
- Small screwdrivers (Phillips and slotted)
- Small adjustable wrench
- Scissors
- X-acto knife
- Ruler
- Small file for smoothing circuitboard edges
- Masking tape
- Pen (for writing on masking tape)
- Multimeter
- Cardboard (scrap is fine)
- Magnifier may be useful
- Extension cord/power strip may be helpful at the workshops

Kit Contents

Level 1 Kit

Anti-Static Bag

Item	Qty	Vendor	Part
Fairchild QRB1134 light sensor	2	Digi-Key	QRB1134
ARC 1.1 PCB	1	Barello.net	ARC-01
ATMEGA16-16PC microcontroller	1	Digi-Key	ATMEGA16-16PC-ND
TI 754410 H-bridge	1	Digi-Key	296-9911-5-ND
RS232 driver	1	Future Electronics	SP232ACP
LM2940CT-5.0 voltage regulator	1	Future Electronics	LM2940CT-5.0
ZTT-16.0MX ceramic resonator	1	Digi-Key	X908-ND
red LED	1	Digi-Key	160-1708
green LED	1	Digi-Key	160-1710
47 uf 16v electrolytic capacitor	3	Digi-Key	P11196-ND
.1 uf ceramic Z7T .1" capacitor	8	Digi-Key	P4924-ND
diode, 1A	1	Digi-Key	1N4001GCT-ND
180 ohm 1/8 W resistor (brown-gray-brown)	3	Digi-Key	180EBK-ND
330 ohm 1/8 W resistor (orange-orange-brown)	3	Digi-Key	330EBK-ND
680 ohm 1/8 W resistor (blue-gray-brown)	2	Digi-Key	680EBK-ND
1k ohm 1/8 W resistor (brown-black-red)	2	Digi-Key	1.0KEBK-ND
4.7k ohm 1/8 W resistor (yellow-violet-red)	3	Digi-Key	4.7KEBK-ND

Electro-Mechanical Bag

Item	Qty	Vendor	Part
40 pin DIP socket	1	Jameco	112310
16 pin DIP socket	2	Jameco	112221
DB25 solder cup male	1	Jameco	15114
DB9 solder cup female	1	Jameco	15771
10-contact IDC socket connector	2	Jameco	32491
1x3 .1" boardmount socket	2	Samtec	SSW-103-01-T-S
1x4 .1" boardmount socket	3	Samtec	SSW-104-01-T-S
1x3 .1" male header	3	Jameco	103341
2x3 .1" male header	2	Jameco	115035
2x5 .1" male header	1	Jameco	67820
2x5 .1" right angle male header	1	Jameco	203932
1x3 .1" connector housing	2	Jameco	157382
female crimp pin	7	Jameco	100765
switch, SPDT, PCB mount	1	Digi-Key	EG1903-ND
.1" shorting jumper	1	Digi-Key	S9002-ND
shrink tubing, 3/32" x 1/2"	2	Digi-Key	FP332K-R50-ND
cable tie	1	Jameco	126543
velcro, 1/2" x 2" loop, adhesive-back	1	McMaster-Carr	9273K173
mounting square (double-stick square)	2	Make An Impression	

Hardware Bag

Item	Qty	Vendor	Part
6-32 x 1.5" standoff (F-F 1/4" OD)	3	McMaster-Carr	91780A337
4-40 x 1" standoff (F-F 1/4" OD)	4	McMaster-Carr	91780A167
4-40 x 1/2" standoff (F-F 3/16" OD)	4	Digi-Key	1893K
4-40 x 1/4" round standoff (F-F 1/4" OD)	2	Digi-Key	3478K
6-32 x 3/8" machine screw	6	McMaster-Carr	90272A146
4-40 x 1" machine screw	1	McMaster-Carr	90272A115
4-40 x 1/2" machine screw	4	Digi-Key	H346
4-40 x 1/4" machine screw	18	McMaster-Carr	90272A106

2-56 x 1" machine screw	4	McMaster-Carr	91772A086
small wood screw (in motor if not here)	2		
4-40 nut	5	McMaster-Carr	90480A005
2-56 nut	4	McMaster-Carr	90480A003
#4 flat washer	8	McMaster-Carr	92141A005
#4 lock washer	2	McMaster-Carr	91113A005
#2 lock washer	8	McMaster-Carr	91113A003

Loose

Item	Qty	Vendor	Part
chassis	1	Pololu	
Solarbotics GM8 motor	2	Solarbotics	GM8PW
wheel	2		
rubber band (for wheel)	4		
Tamiya ball caster (pair)	1	Pololu	0066
battery holder	1	Jameco	216215
velcro, 1/2" x 16" strap	1	McMaster-Carr	94905K33
4-conductor cable, 3'	1	Jameco	31860
10-conductor ribbon cable, 3'	1	Jameco	135538
servo cable	2	Barello.net	
insulated copper wire, 7"	2	McMaster-Carr	8073K64

ARC Board Connection Kit

This optional add-on kit provides parts to terminate sensor wires and align them in housings for easy connection to the board sockets. The default kit has wires tinned and inserted directly into the sockets.

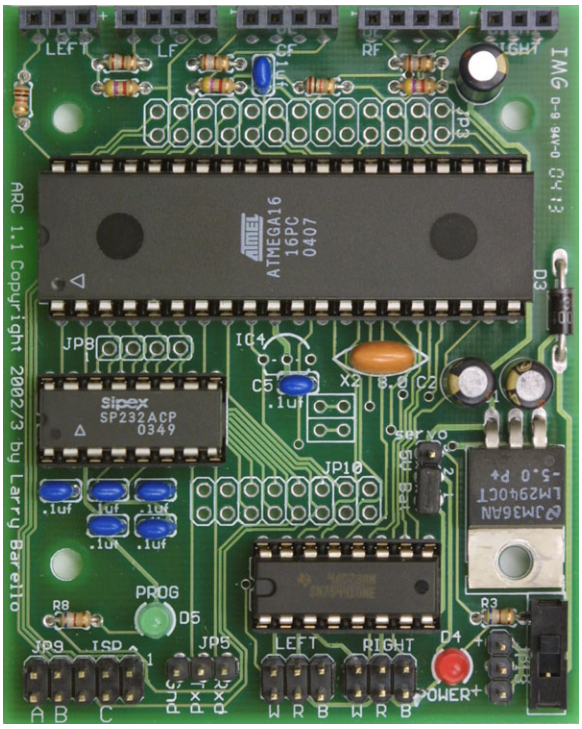
Item	Qty	Vendor	Part
1x3 .1" pin housing	2	Jameco	157382
1x4 .1" pin housing	3	Jameco	100802
male crimp pin	20	Jameco	145357

Vendor Information

Digi-Key	http://www.digikey.com/
Jameco	http://www.jameco.com/
Barello.net	http://www.Barello.net/Commerce/
McMaster-Carr	http://www.mcmaster.com/
Pololu	http://www.pololu.com/
Solarbotics	http://www.solarbotics.com/
Samtec	http://www.samtec.com/
Future Electronics	http://www.future-active.com/
Mark III Robot Store	http://www.junun.org/MarkIII/
Make An Impression	http://www.makeanimpression.net/
B.G. Micro	http://www.bgmicro.com/
All Electronics	http://www.allelectronics.com/

AVR Robot Controller 1.1 Assembly Instructions

Preparation



Identify the parts used in the AVR Robot Controller (ARC). Refer to the Kit Contents section for a list of parts in each bag.

- Anti-Static Bag: everything except the QRB1134 light sensors.
- Electro-Mechanical Bag: everything except the velcro, mounting squares, shrink tubing, and rubber bands.
- Loose items: the battery holder, 4-conductor cable, and ribbon cable.

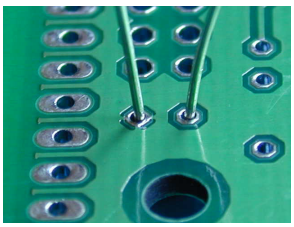
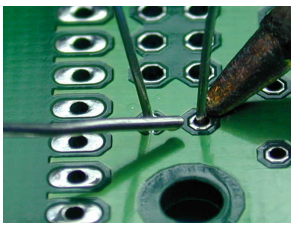
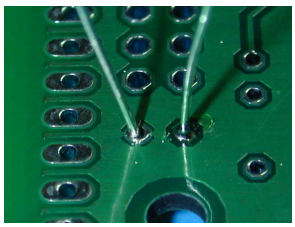
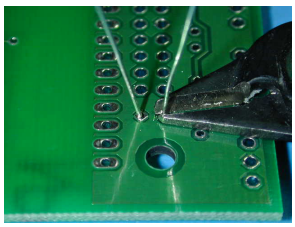
It is recommended that you return unused items to their original locations so that you can find them later!

Soldering Overview

The ARC can be assembled using a fine tip 25-40wt soldering iron, thin 60:40 rosin core solder, flush side cutters and a pair of small needle nose pliers. Additional items that are useful are a solder tip cleaner (damp sponge or cloth), paste solder flux and a small gauge wire stripper (18-26). Because it is fairly small you need a well-lighted area to work in. A magnifying glass might be useful for inspecting solder joints.

The side cutters are used to trim component leads flush to the board after soldering them. With a little practice the side cutters can be used to strip wire as well: the trick is to cut into the insulation, but not the wire, and then pull the end off.

The key to obtaining a good solder joint is to get everything hot enough so that the solder “wets” the surface and wicks into the spaces between the wire and what you are soldering it to. First clean the iron tip by wiping it across a damp sponge. Then tin the tip with a bit of fresh solder. Then place the tip against the pin and the solder pad, i.e. wedging it in the corner of the two. Then touch the solder to the pin or the board near to the tip. Usually it takes just a fraction of a second to get the joint hot enough to melt the solder. Then leave the tip on the joint for a couple seconds more until the solder visibly wicks into the hole. If the hole looks empty you can put another touch of solder on. When done, snip the excess lead material off flush to the bottom of the board. Usually cleaning and tinning the tip is needed only between rounds of soldering or after soldering 20-30 joints in a row.

You need a workspace as well. The ARC uses static-sensitive components and you need to exercise care while assembling it. The best is an anti-static mat with a wrist strap and grounding it to the same ground as the soldering iron uses. An alternative is to work on a bare wooden surface. Wood neither conducts nor accumulates static electricity. Sometimes you can obtain a sheet of conductive foam and work on that. Keep it grounded through a large resistor (e.g.1-2 meg ohms) and make sure some exposed skin touches the surface before you pick up static sensitive parts (anything that looks like an IC). Once the board is assembled and the IC’s inserted, it is much less sensitive to static discharge.

Prepare Boards

The PCB comes with a serial adapter and a programming adapter that need to be detached. First break the long line of holes. You might find it easier to break if you run a razor knife along the top and bottom of the line of holes. Then snap the two smaller boards apart. Finally, if you want, you can sand or file the rough edges smooth.

Controller

The control board has been designed with oversized holes to facilitate hand assembly and disassembly. The problem is that components will fall out when the board is turned upside down. If you assemble from shortest to tallest components, then it is possible to load several parts onto the board, cover it with a piece of conductive foam or cardboard and turn it over. Still, parts will tilt and move about. The trick is to solder one pin of each part into place, turn the board over and re-position the parts by re-heating the soldered pin while pushing on the other side with your finger. When all the parts are positioned to your satisfaction, then turn the board back over and finish soldering the rest of the pins.

The picture on the right shows the completed ARC board for the SRS Level 1 Robot Kit. The "front" of the board is at the top of the picture. This refers to how the board will be mounted on the robot, and is used to describe locations in the instructions below.

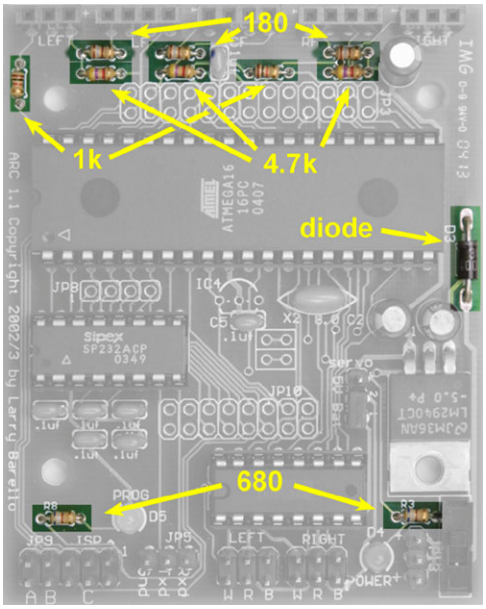
Solder Components

Part locations are labeled on the board, but some labels are incomplete due to solder pad location, and some parts have changed. ***Please read the instructions to ensure that parts are installed correctly.***

The following list is in order of shortest to tallest component. All the components in each section can be soldered at one time, or each component can be soldered in separately.

- **Resistors and diode**

  The picture on the right shows the locations for:
  - 3 - 180 ohm resistors (brown-gray-brown)
  - 3 - 4.7k ohm resistors (yellow-violet-red)
  - 2 - 1k ohm resistors (brown-black-red)
  - 2 - 680 ohm resistors (blue-gray-brown)
  - 1 - diode

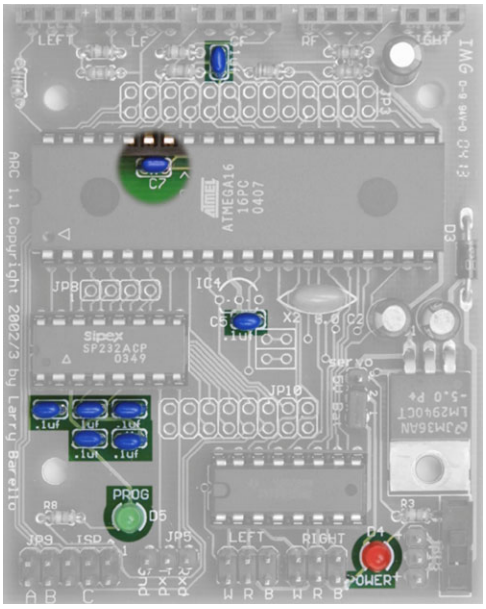
  

  Important: Note the locations for the 180 and 4.7k resistors at the top of the picture. They are installed in locations marked 680 and 33k, respectively. (R3 and R8 remain 680 ohms, as shown at the bottom of the picture.) These resistors are selected to work well with the Fairchild QRB1134 phototransistors (light sensors) in the kit. For more information, read the section titled "Left, Center, and Right Floor Sensor Inputs."

  Orientation of the resistors is not important. Resistors do not have "polarity," so it does not matter which end goes into each hole on the board.

  The diode is inserted with the band toward the end of the arrow on the board. If you know what you are doing and want maximum power you may omit the diode, replacing it with a short piece of wire. This will remove the reverse battery protection that prevents destruction of the H-Bridge chip and/or attached servos. The voltage regulator protects the rest of the board from battery reversal.

- **Small capacitors and LEDs**



The picture on the right shows the locations for:

- 8 - 0.1uF capacitors
- 1 - red LED
- 1 - green LED

Orientation of the small capacitors is not important.

Note that one of the capacitors (second from the top) will be under the microcontroller. Be sure that it sticks up no higher than the DIP socket (to be installed in the next step). You may need to bend the capacitor's leads a bit to get it to sit close enough to the board. Another option is to mount the capacitor "lying down" by gently putting right angle bends into the leads before inserting them into the board.

The red LED is power and the green LED is program. The LEDs have a flat side that is oriented toward the near edge of the board (the edge at the bottom of the picture). There is a long and a short lead; the short lead should be near the edge of the board and the longer lead toward the center of the board.

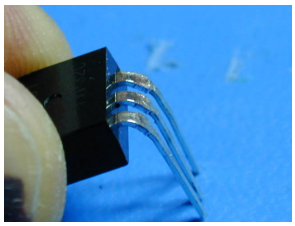
- **DIP sockets and voltage regulator**

In this step you will install:

- 1 - 40-pin DIP socket
- 2 - 16-pin DIP sockets
- 1 - LM2940 voltage regulator

All ICs can be mounted directly on the board. However, if you are a beginner or an experimenter, you probably should use the DIP sockets so the ICs are easy to replace. Make sure the notch on the bottom of the socket matches the notch in the markings on the board (on the left side in the photos).

If you want to maximize the power handling capabilities of the 754410 H-bridge, you must solder that chip directly onto the board. The ground plane of the board provides the heat sink needed for maximum power output. This is not recommended for beginners.

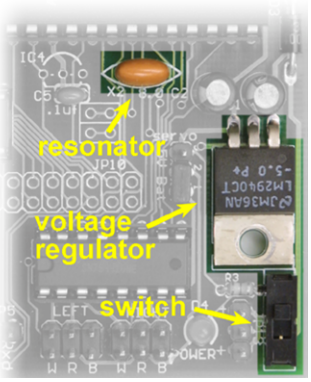


Bend the leads of the voltage regulator half-way through the fat part, near the body of the IC, as shown in the picture on the right. This will align the mounting hole over the hole in the PCB and the IC should lie flat when soldered in. (Board location is shown in the picture in the next section.)

- **Switch and resonator**

In this step you will install:

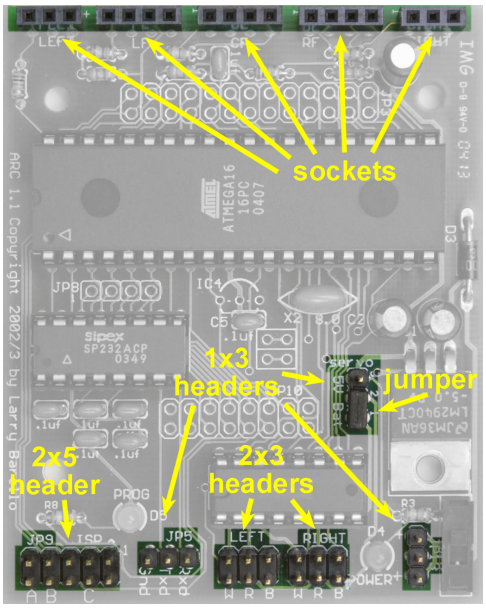
- 1 - switch
- 1 - ZTT-16.0 resonator



Install in the locations identified in the picture on the right (which shows the lower right corner for the board). Orientation is not important for the switch or resonator. The resonator is optional. The ATmega16 has an internal 1, 2, 4, or 8 MHz oscillator. In addition, it can operate with an external source of up to 16MHz, which this resonator provides.

- **Sockets, headers, and jumper**

  The picture on the right shows the locations for:
  - 2 - 1x3 boardmount sockets
  - 3 - 1x4 boardmount sockets
  - 3 - 1x3 male headers
  - 2 - 2x3 male headers
  - 1 - 2x5 male header
  - 1 - shorting jumper



  Install the sockets along on the front of the board (as shown at the top of the picture). The headers should be installed with the shorter end of the pins inserted into the board. Install the 1x3 pin headers in the locations marked JP5 (serial communications), JP13 (power), and "servo." Install the 2x3 pin headers (motors) at the back of the board, in the locations marked LEFT and RIGHT. Install the 2x5 pin header (programming) in the location marked JP9.

  Install the shorting jumper on the "servo" header. The picture on the right shows the jumper positioned to select "Bat." The other option is "5V." Either setting is fine for Level 1. Please see "Left and Right Motor Connectors" in the ARC 1.1 Hardware Description section for details.

- **Large capacitors**

  In this step you will install:
  - 3 - 47uF capacitors

  Pay attention to the orientation of these devices. The longer lead goes into the hole marked with a '+'. The gold bar on the side of the capacitor should be on the side marked with a '-' sign.

Cleaning and Inspection

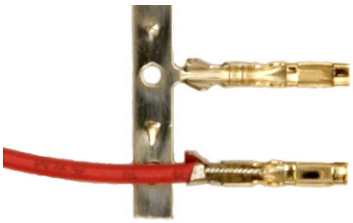
After all components are installed, inspect the bottom of the board with a magnifying glass. Look at each solder pad and make sure that the solder is shiny and has a "wet" look. Reheating and applying a touch of solder can rework pads that are incompletely filled with solder or look gray and dull.

Look for solder bridges between pads or pins that are close together. If you used a fine tip iron and no more solder than needed for a good joint, you shouldn't have any bridges.

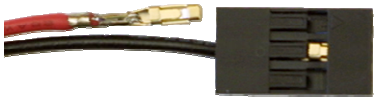
You can clean the flux (from the solder) from the board with denatured ethanol (alcohol). Use an old toothbrush to scrub all the pads with the alcohol.

Attach Connector to Battery Holder

For this step, you will need:
- 1 - battery holder
- 2 - female crimp pins
- 1 - 1x3 connector housing



The ends of the battery holder's wires are already stripped and tinned. Place one wire into the "trough" of a female crimp pin, with the insulation just barely reaching into the area between the two triangles. Using needle-nose pliers, fold over one triangle to hold the wire in place for soldering (as shown in the picture on the right). Apply a *small* amount of solder between the bare wire and pin, then crimp the pin closed using needle-nose pliers or a specially-designed crimp tool. Repeat with the other wire.



Insert the crimp pins into the housing with the flat part of the pin against the flat part of the connector. The black wire must go into the middle location of the housing; the red wire can go on either side.

Install Chips

Be sure to inspect the board before applying power. Connect the battery pack to the 3-pin header next to the switch (JP13). With the switch in the on position (toward the near edge of the board) the red power LED should light.

Now you can install the various chips. (Turn off power to the board first.) All chips are installed with the notch (pin 1) oriented away from the switch and voltage regulator (the notch is on the left side in the photos above). The RS232 driver is installed in the location closer to the ATmega16; the 754410 H-bridge is near the 2x3 headers.

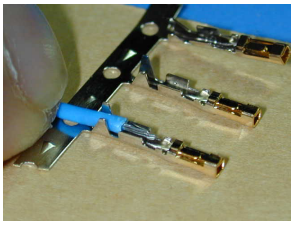
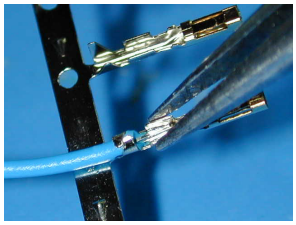
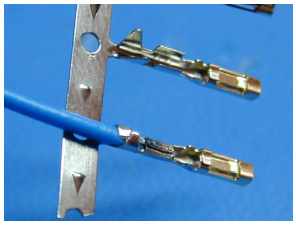
Serial Cable

The kit includes materials to make a serial interface cable that can be used to connect the ARC board to your PC. This is useful when debugging your programs as it allows the program to tell you what is going on.

For this step, you will need:

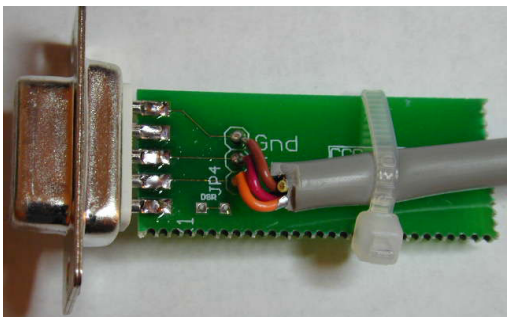
- 1 - serial adapter PCB (the one with “Gnd Tx Rx” labeling)
- 1 - DB9 solder cup
- 3’ - 4-conductor cable
- 3 - female crimp pins
- 1 - 1x3 connector housing
- 1 - cable tie

- ❑ Strip one inch of gray insulation from each end of the 4-conductor cable to expose the wires. Snip off the bare wire and one of the four insulated wires.
- ❑ Strip 1/4” of insulation from the ends of each remaining wire and tin the exposed ends.
- ❑ Trim the tinned wire to 1/8” long.
- ❑ Solder to the crimp pin sockets and crimp the pins to the wire, as shown in the pictures below.

- ❑ Insert the crimped pins into the connector housing. The pins should snap into place when inserted correctly.
- ❑ Mark the 3-pin socket with Rx, Tx, and GND, as shown in the picture on the right. A piece of masking tape (not included) works well. Each line will have a corresponding label on the PCB.
- ❑ Solder the DB9 connector to the board, paying attention to the orientation so each solder cup covers a pad.
- ❑ Solder the wires to the board, referring to the socket connector to determine which color wire to use for each signal.

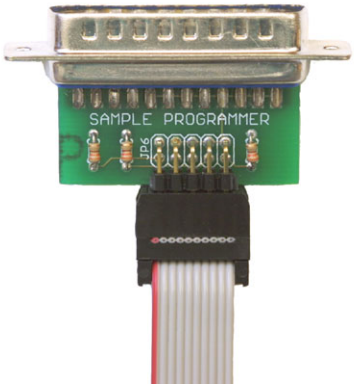
- ❑ Fasten the cable to the board using the cable tie. This provides strain relief.



Finished Serial Connector

Programming Cable

The kit is supplied with material to make a programming cable that works with the BASCOM Basic Compiler or the AVRDUDE program (included in the WinAVR suite for C/C++ programming). BASCOM and AVRDUDE work with any third party programmer that has an Atmel 10-pin programming socket, so you only need to build the cable if you don't already own a programmer.



The programmer uses a male DB25 connector that plugs directly into a PC printer (parallel) port.

For this step, you will need:

- 1 - programming adapter PCB
- 3 - 330 ohm resistors (orange-orange-brown)
- 1 - 2x5 pin ***right angle*** header
- 1 - DB25 solder cup
- 3' - 10-conductor ribbon cable
- 2 - IDC socket connectors (and strain-relief clips)

Solder the resistors, header, and DB25 connector to the board.

Hold an IDC connector with the sockets aiming down and the "pin 1" arrow on the right side. Align the ribbon cable with the red stripe toward you, and slide the cable into the connector from the right side until the edge of the cable just aligns with the left side of the connector. The red stripe should be aligned with the arrow. Gently press on the top of the connector to crimp it together. You may find it useful to use a vise or pliers, but be careful, squeeze gently, and try to apply even pressure. After the connector is together, fold the cable over the top so that it extends to the left. Insert a strain-relief clip into the top. The final result should look like the picture on the right.

Install the second IDC connector on the other end of the cable in the same way. Plug one end of the cable into the programming adapter, aligning the red stripe with pin 1, which is next to the "JP6" label. The other end plugs into the programming header on the ARC board, with the red stripe aligned with the caret by pin 1.

AVR Robot Controller 1.1 Software Tools

This is a good point to test your robot controller. This section explains the basic terminology and procedure for programming, and has instructions for loading your microcontroller with a program that will flash the green LED.

There are three main steps in teaching your robot to do something:
1. Write instructions (a "program") for the robot to follow.
2. Translate ("compile") those instructions into binary code that the robot can execute.
3. Convey ("download") the binary code to the robot's microcontroller using a special cable (the "programmer").

Options for the first two steps are discussed below in the Tool Options section.

To download your program, you need to connect your computer to your robot using one of the following options:

- The least expensive option is to use the programmer included as part of the kit. It can be used for either BASIC or C/C++ on Windows if you have a (DB-25) parallel port on your machine. (Note that it will most likely not work to use a USB-to-parallel converter; they do not provide full parallel port functionality, just what's necessary for printing!)
- If you are using a Mac, or a Windows machine without a parallel port, you can purchase an AVR-ISP programmer from Digi-Key (part number ATAVRISP-ND) for $29. That programmer hooks up to a (DB-9) serial port. You will also need to get a USB-to-serial converter if your machine does not have a serial port.

Tool Options

Free, high-quality tools are available for the Atmel AVR series of microcontrollers. On Windows systems, two popular options are the BASCOM (BASIC) compiler and the WinAVR suite of tools for C/C++ programming. Mac OS X supports the avr-gcc C/C++ compiler (which should work equally well on Linux and FreeBSD).

The following sections each provide instructions for installing a compiler and downloading a simple program to your ARC board. Choose the option that best suits your needs and follow the steps in that section.

BASCOM on Windows

BASCOM-AVR is an "integrated development environment" (IDE) that provides an editor, BASIC compiler, and downloader. This is a good starting point for people new to programming.

You can download the BASCOM-AVR DEMO from http://www.mcselec.com. This site also provides information on installing BASCOM.

Copy the "BASCOM Samples" folder from http://www.seattlerobotics.org/WorkshopRobot/Level1/ (or the Workshop CD's "Samples" folder) to a convenient place on your hard drive.

Start the BASCOM-AVR program. Use File/Open to open FlashLED.bas in the "01aFlashLed" folder (in BASCOM Samples).

While power to the ARC board is off, plug the programming cable into the ISP header on the board and the parallel (printer) port of a PC. (If you have an AVRISP or STK200/300 compatible programming cable you can use that with BASCOM by changing the programmer in the Options/Programmer menu. (The kit's programmer uses the default "Sample Electronics programmer" option.) Please refer to the BASCOM documentation for additional information.)

Turn on power to the ARC board, click on the Compile button (or use menu Program/Compile), then click on the Run button (Program/Send To Chip). The Sample programmer should pop up with the ARC CPU identity listed (m16). Click on the Auto-Program button (menu: Chip/Autoprogram) and the program should load in a couple seconds. Close the Sample programmer window.

The green program LED flashes with a one second cycle time after a successful download. If the LED flashes very slowly, then the oscillator options for the CPU have not been set and the CPU is likely running off the factory default 1 MHz internal oscillator. You can re-program the chip to run off the external crystal/resonator, or the internal 8 MHz oscillator by reading about the fuse bits and programming them appropriately. See "Customizing the Microcontroller" below.

WinAVR (avr-gcc) on Windows

WinAVR (pronounced "whenever") is a suite of software development tools for the Atmel AVR series of microprocessors. It includes the avr-gcc compiler for C and C++, AVRDUDE downloader, and Programmers Notepad editor.

You can download the WinAVR suite from http://sourceforge.net/projects/winavr/. When you run the installer, it is a good idea to **uncheck the option "Add Directories to PATH."** (Adding the WinAVR tools to the path may cause grief for other apps since that puts Cygwin (Linux emulation) files before system files.)

If you are using an "NT" version of Windows (e.g. NT 4.0, 2000, XP, 2003), you will need to do the following to set up AVRDUDE (it enables communication with the ports needed for downloading):

- Launch a Command Prompt (from the Programs menu, choose Accessories, then Command Prompt).
- At the prompt, type: `cd c:\WinAVR\bin` and hit Enter
- Then type: `install_giveio` and hit Enter
- You should see output from this program, ending with a "Success" message. Close the command prompt window.

Copy the "C Samples" folder from http://www.seattlerobotics.org/WorkshopRobot/Level1/ (or the Workshop CD's "Samples" folder) to a convenient place on your hard drive.

Double-click on "**LaunchPN.bat**" in the "C Samples" folder. This will launch Programmers Notepad with the path changes needed by the compiler and downloader. (If you installed WinAVR in a non-default location, you'll need to edit this batch file.)

While power to the ARC board is off, plug the programming cable into the ISP header on the board and the parallel (printer) port of a PC. (If you are using an AVRISP via a serial (or USB) port, you will need to open the "makefile" file in the "01a FlashLED" folder and change two definitions: "AVRDUDE_PROGRAMMER = avrisp" and "AVRDUDE_PORT = com1". You may also want to define environment variables for these.)

In Programmers Notepad, use File/Open to open FlashLED.c in the "01a FlashLED" folder (in C Samples). In the Tools menu, choose the "Make All" option. An "output" window will show the results of your build. (If you get a message that "the system cannot find the file specified," you probably just need to restart Programmers Notepad using LaunchPN.bat.) To download the program, turn on power to the ARC board, then from the Tools menu, choose the "Program" option.

You should now see the green LED flash with a one-second cycle time.

avr-gcc on Mac OS X

The easiest option for programming on the Mac is to use the C compiler and UISP downloader available via a package installer from http://www.eecs.berkeley.edu/~mseeman/resources/macmicro.html. It's an easy install process, but only compiles C, not C++, programs.

If you would like to use C++, you'll need to do a full manual install (building all of the tools from source). Fortunately, there are complete instructions in the AVR runtime library manual (avr-libc-user-manual-1.0.4.pdf) available at http://savannah.nongnu.org/projects/avr-libc/. Follow the steps in section 7.8, "Installing the GNU Tool Chain." This document also lists the locations for downloading the other tools. One thing to be aware of is that you need to have TeX installed to build AVRDUDE (it's apparently used when building the included documentation); if you have Fink installed, use the command: "fink install tetex". You can avoid this by using the UISP downloader instead of AVRDUDE.

Note: The AVR runtime library manual mentioned in the previous paragraph contains lots of useful information for programming the Atmel ATmega processors, making it a good resource regardless of which compiler option you choose.

Using these tools requires using the command line interface to the Mac's Unix-based underpinnings. If you don't know how to use the Terminal program, you should read up on that before tackling the install and use of the AVR tools.

Copy the "C Samples" folder from http://www.seattlerobotics.org/WorkshopRobot/Level1/ (or the Workshop CD's "Samples" folder) to a convenient place on your hard drive.

While power to the ARC board is off, plug the AVRISP device into the ISP header on the board and connect it to a USB-to-serial converter attached to your Mac. Open the makefile in the "01a FlashLED" folder and change two definitions: "AVRDUDE_PROGRAMMER = avrisp" and "AVRDUDE_PORT = ___". For the port, you will need to use the name of the serial port supplied by your USB-to-serial adapter. You can get a list of available serial devices by typing in the Terminal window: "`ls /dev/cu.*`". You will need to supply the full name, e.g. "AVRDUDE_PORT = /dev/cu.USA281b1P2.1".

In the Terminal window, change to the "01a FlashLED" folder. At the command line type "make" to compile the program. Turn on power to the ARC board, then type "make program" to download the program.

You should now see the green LED flash with a one-second cycle time.

Customizing the Microcontroller

The ATmega16 microcontroller provided in the kit can be customized by changing "fuse bits." These settings may be modified with built in programmer that comes with BASCOM, or with the AVRDUDE program included in the WinAVR suite.

Every fuse bit has a default value. There are several fuse states that it's useful to change. These are explained in more detail in the "ATmega16 Overview" section. The suggested changes are: (1) set the internal oscillator (clock)

to 8 MHz; (2) enable brown-out detection; (3) disable JTAG; and (4) disable erasing EEPROM when downloading a new program. Please refer to the chip data sheet (www.atmel.com) for a complete list of the fuse bits.

The following steps set the microcontroller to the suggested state.

If you are using BASCOM-AVR: Connect the programmer to your board as above, select "Send to Chip" from the Program menu to bring up the Sample programmer window. Click on the "Lock and Fuse Bits" tab.

In the Fusebits section:
Click on Fusebit B and change the option to "0:BODEN enabled"
Click on FusebitA987 and change the option to "0100:Internal RC oscillator 8MHz"
Click on the WriteFS button.
In the Fusebits High section:
Click on Fusebit J and change the option to "1:Disable JTAG"
Click on Fusebit G and change the option to "0:Preserve EEPROM when chip erase"
Click on the WriteFSH button.
Close the Sample programmer window.

If you are using avr-gcc and AVRDUDE, downloading the FlashLED (or FlashLEDAlias) program will set the fuse bits as part of the download process. (If you are setting the fuse bits manually, the changes above are accomplished with 0xA4 for the low byte and 0xD1 for the high byte.)

Resources

You can find tools, support, and samples for Atmel's AVR microcontrollers at the following sites:
http://www.seattlerobotics.org/WorkshopRobot/Level1/
Home page for the SRS Robot Level 1 Kit with links for sample code, data sheets, etc.

http://www.barello.net/ARC
AVR Robot Controller resource page, user guide and sample code.

http://www.atmel.com/atmel/products/prod23.htm
Assembler, Simulator, Application notes, software.

http://groups.yahoo.com/group/AVRRobotControl
Technical support, applications, code fragments, documentation, design files.

http://www.mcselec.com
The BASCOM compiler and various application notes can be found here.

http://www.avrfreaks.net
Discussion groups, sample code and projects, avr-gcc C compiler.

http://sourceforge.net/projects/winavr/
Download site for the WinAVR suite of development tools, including GNU avr-gcc C/C++ compiler and AVRDUDE downloader.

http://winavr.sourceforge.net/
Home page for the WinAVR suite of tools.

http://www.eecs.berkeley.edu/~mseeman/resources/macmicro.html
Information about the AVR tools on Mac OS X and a download for a binary installer that will give you all the software you need to program AVR chips on Mac OS X in C.

http://savannah.nongnu.org/projects/avr-libc/
Detailed info on the free AVR tools is located on the site for the AVR runtime library.

Chassis Assembly

Preparation

Identify the parts used in the chassis. Refer to the Kit Contents section for a list of parts in each bag.

- Anti-Static Bag: QRB1134 light sensors.
- Electro-Mechanical Bag: velcro, mounting squares, shrink tubing, and rubber bands.
- Hardware Bag: everything except the four 1" standoffs and eight of the 4-40 x 1/4" machine screws.
- Loose items: everything except the 4-conductor cable and ribbon cable.

Remove the paper from the (twelve) chassis pieces. (The Level 1 kit contains the chassis parts needed for all levels; keep the extra pieces somewhere convenient for use later.)

Cut the 1/2" x 2" strip of adhesive-back velcro loop into two 1/4" x 2" strips. Attach one of them to one of the long sides of the battery holder. This will help hold the batteries in place. Attaching the second strip to the other side is optional; it will make the holder even more secure, but it will be harder to insert and remove it.

Cut each 1/2" piece of shrink tubing in half. Slide one 1/4" piece onto the orange wire of a servo cable (you can gently pull to separate the colored wires, but don't shorten the cable). Referring to the picture on the right, solder the orange wire to the "left" connection on the motor, keeping the shrink tubing away from the joint while you work. Slide the tubing over the connection and shrink it with a heat gun or hair dryer. Repeat with the red wire, and then for the second motor.

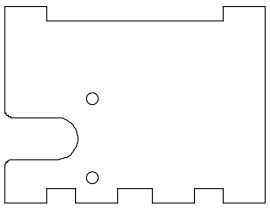
Trim the wires connected to the light sensors, leaving 6" of colored wire. Strip the insulation off 1/4" of the end of each wire, then tin those ends. *Note*: if you are using the optional ARC Board Connection Kit, connect male crimp pins to the light sensor wires and insert them into a 4-position housing with the orange wire in the slot with the triangle, and the other wires arranged so that the color order is orange - green - blue - white.

Using leftover wires from the light sensors, cut two 7" lengths of blue and two of white. Strip and tin ~1/4" on one end and 3/8" on the other end. (Do not use ARC Board Connection Kit parts here.)

Motors

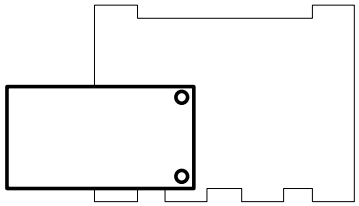
Find the two copies of the "motor plate" pictured on the right.
Hardware:

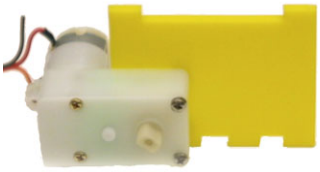
4 2-56 x 1" screws
4 2-56 nuts
8 2-56 lock washers



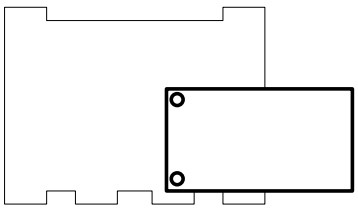
Place a motor in front of the plate, with the holes aligned and the motor's *green* axle in the circular cutout (as shown on the right).

Put a lock washer onto a screw and insert the screw through the motor and then through the hole in the plate. Finish with another lock washer and a nut, but don't tighten it, yet. The assembly should stack like this: screw head, washer, motor, plate, washer, nut. Repeat the same sequence with the other hole, then tighten both.





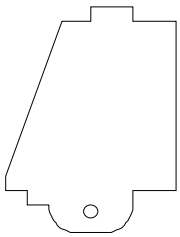
For the other motor, hold the other piece with the cutout on the right and the motor placed as shown in the picture on the right. Fasten in the same manner as the first motor.
Stretch a band around each wheel, but don't connect them to the motors, yet.



Light Sensors

Find the two copies of the "light sensor plate" pictured on the right.
Hardware:

1 4-40 x 1" screw
1 4-40 nut
2 4-40 lock washers
4 4-40 flat washers



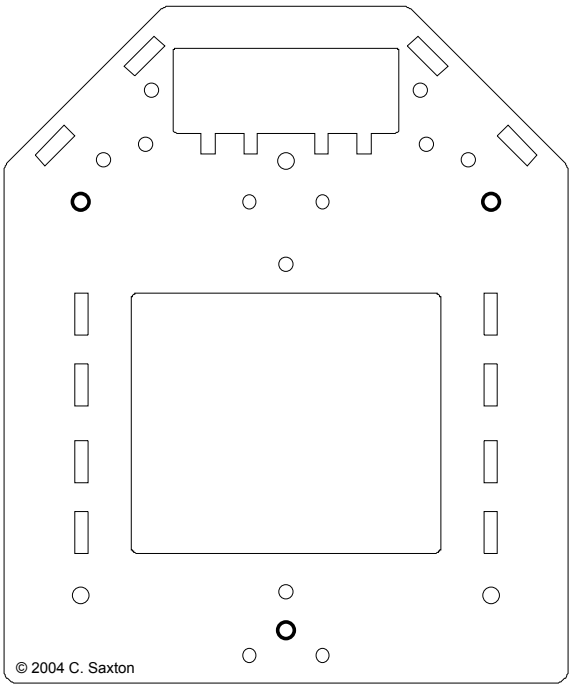
Stack items onto the screw as follows: lock washer, plate, 2 flat washers, both light sensors (orient them the same way), 2 flat washers, the other plate (oriented the same direction as the first plate), lock washer, nut. Tighten.

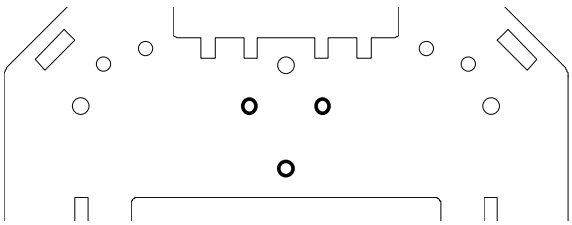
Bottom Plate

The picture below shows the bottom plate.
Hardware:

3 6-32 x 1 1/2" standoffs
3 6-32 x 3/8" screws

Attach the standoffs to the plate through the highlighted holes, with the screws on the side with the etching.





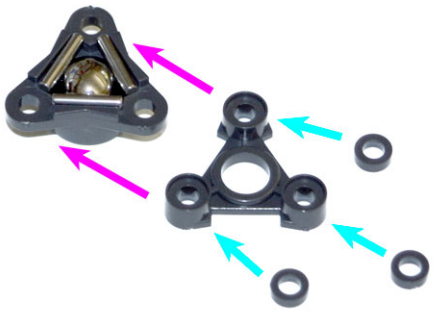
Open the Tamiya ball caster package and retrieve the following parts:

1 H3 (ball holder, triangular)
1 ball
3 shafts
1 H2 (triangular)
3 H1 (spacers)
3 3x15mm machine screws (longer ones)
3 nuts

Note that the package contains additional parts (for a second caster and alternate configurations), but they are not needed here.
Insert the ball into H3, and place the shafts on top of the ball, parallel with the edges of the holder, as shown in the upper left image in the picture below.

Place H2 on top of the H3 assembly, as shown by the pink arrows.

Place the three H1 spacers into the indentations on H2, as shown by the blue arrows.

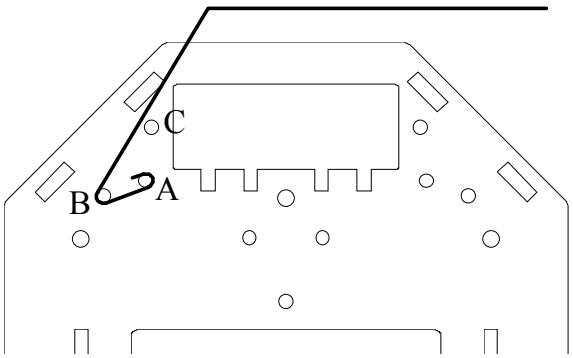


Insert each machine screw from the "ball" side, up through H3, H2, and a spacer. Hold the bottom plate of the chassis with the standoffs on the top and place it over the caster, inserting the screws through the holes highlighted in the picture on the left. Fasten with the nuts.

Strip about 2” from one end of each 7” length of 18 AWG insulated wire.
Hardware (for one side):

1 4-40 x 1/4” round standoff
1 4-40 x 1/4” screw
2 4-40 x 1/2” screws
2 4-40 washers
2 4-40 nuts

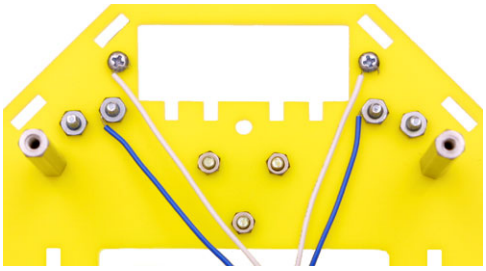
Mount the standoff in hole C, on the same side of the plate as the caster, using the 1/4” screw.



For holes A and B, put a washer on a 1/2” screw and put the screw through the plate from the side with the caster. Attach a nut (on the side with the standoffs). Bend the stripped end of the wire around pins A and B as shown. Fasten between the washer and the plate. Make sure that the section of the wire next to C is stripped. Place additional bends in the wire to make an effective bump sensor.

Hold the plate so that the caster is on the bottom.

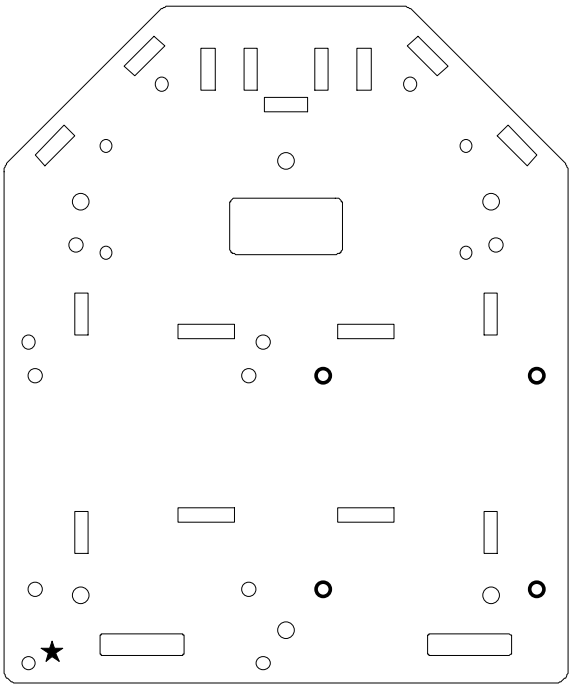
Connect one of the spare blue wires (from the light sensors) to A, wrapping the 3/8” tinned end around the screw and fastening it under the nut (on the opposite side of the chassis from the bumper wire). Connect one of the spare white wires to C, fastening it under the screw head (on the same side of the chassis as the blue wire).



Repeat the bumper installation on the other side.

Top Plate

The picture at the right shows the top plate as seen from above.
Note: this piece is not symmetric. Make sure that you are holding the piece so that there’s a hole in the lower left corner (marked by the star in the picture). If the hole is in the lower right corner, flip the piece over.



Hardware:

4 4-40 x 1/2” standoffs
4 4-40 x 1/4” screws

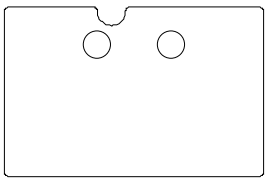
Insert the screws through the highlighted holes *from underneath, sticking up through the plate*.

Attach the standoffs to the screws.

Double-check to make sure that with the standoffs sticking up from the plate, you have a hole in the lower left corner.

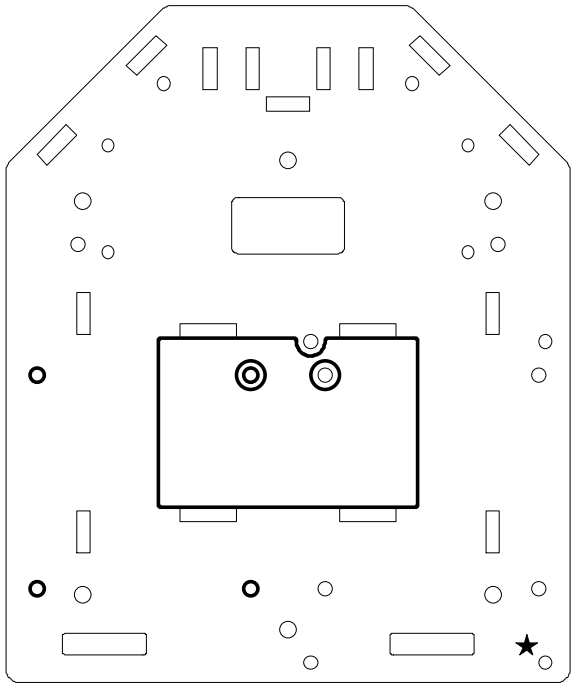
Flip the top plate over. The screws will be visible, and the standoffs will be underneath (and you should have a hole in the lower right corner).

Find this piece (the battery spacer), and hold it like this:



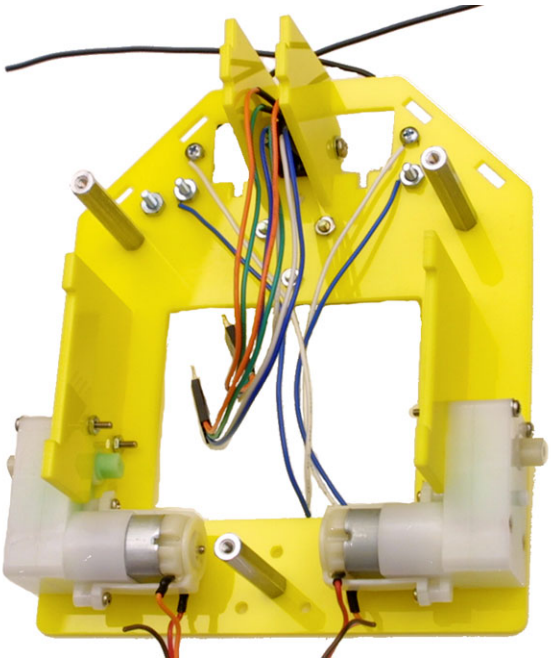
You will mount this to what is the underside of the top plate (which is now face up since you flipped it over!)
Stick the two mounting squares onto the battery spacer. Then, flip the spacer over and attach it to the top plate, as shown on the far right. The battery spacer will fit between the slots and the holes will be aligned.

Cut the velcro strap in half (into two 8” strips). Feed each strip through a pair of slots so that the “hook” side of the velcro is against the chassis and the ends are on the side with the battery spacer, as shown on the right.



Final Assembly

To assemble the robot:
Put the bottom plate on the ground with the caster down and the standoffs up.



Place the light sensor assembly into the slots at the front of the robot, as indicated at the top of the picture on the right. The angled edge will go toward the interior of the robot.

Place the tabs of the motor assemblies into the slots on the bottom plate, with the motors resting on the bottom plate and oriented as shown in the picture on the right.

Hold the top plate with its standoffs sticking up. Feed the wires from the light sensors and bumpers through the rectangular hole near the front of the top plate. Align the slots in the top plate with the tabs from the motor and light sensor assemblies.

Hardware:

- 3 6-32 x 3/8” machine screws
- 2 small (wood) screws
- 4 4-40 x 1/4” screws

Fasten the 6-32 x 3/8” machine screws through the top plate and into the standoffs.

Attach the wheels to the motors and secure using the small screws.

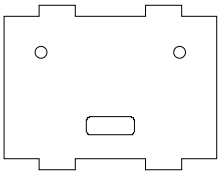
Place the ARC board on the standoffs mounted to the top plate, oriented so that the switch is in the corner. Attach the board to the standoffs using the 4-40 x 1/4” screws.

Leftover Parts

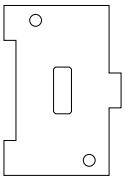
At this point, your robot is together and you're probably wondering about those extra parts. They are used for Levels 2 and 3 of the SRS Robot. Here's a summary.

The hardware bag should still contain four 1" standoffs and eight 1/4" machine screws. They are explained below.

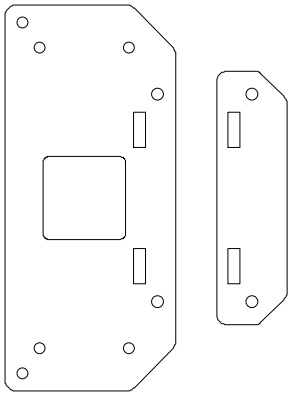
There will be five plastic chassis pieces remaining:



These two pieces are used to hold Sharp GP2D12 infrared distance sensors (part of the Level 2 kit). They mount on the front corners of the chassis between the top and bottom plates.



This piece can hold the Devantech SRF-04 Ultrasonic Rangefinder (sonar distance sensor). It mounts with the single tab into a slot near the front of the top plate and the double-tab side held in place by either of the following pieces (above the top plate).



Either of these pieces can be used as a "cap" for the sonar plate. They are mounted to the top plate via the remaining standoffs and screws (two sets for the small piece, all four sets for the large piece).

The larger piece can hold an LCD screen (part of the Level 2 kit). The LCD can also be mounted directly to the top plate.

Wiring Connections

Plug the servo cables from the motors into the "LEFT" and "RIGHT" pins at the back of the board, using the row of pins further from the edge of the board, with the orange wire aligned with the "W" and the brown wire aligned with the "B."
Plug the wires from the light sensors and bumpers into the sockets on the front of the ARC board as follows:

- Plug the right light sensor wires into the "RF" sockets at the front of the board. There is a "1" identifying position 1. The colors, from position 1 to 4 are orange, green, blue, and white. (If you are using the ARC Board Connection Kit, plug the connector into the board so that the triangle (orange wire) is at position 1.)
- Plug the left light sensor wires into the "LF" sockets, using the same color order as above.
- Plug the right bumper wires into the "RIGHT" sockets at the front of the board. Plug the white wire into position 1; plug the blue wire into position 2. (There is no connection to position 3.)
- Plug the left bumper wires into the "LEFT" sockets, as above.

You may find it useful to label the connectors at the end of the cables for the motors (and light sensors if you are using the connection kit). A small piece of masking tape (not supplied) with "L" or "R" works well.

Place 6 AA batteries into the battery holder, flip the robot over, and fasten the velcro straps around the battery holder to fasten it in place. Feed the connector through the slot in the top plate of the chassis, and plug it into the power connection on the board (next to the switch). Be sure that the housing aligns with the three pins. The red wire can be on either end; the black wire should be in the middle.

ATmega16 Programming

Part 1

Atmel ATmega16 Programming

Part 1

Cathy Saxton
Larry Barello
Jim Wright

Topics

- Atmel ATmega16 has 32 pins that can be used for inputs and outputs
- We're gonna learn how to use them!
- General overview, not language-specific
- Flash green LED
- Read bumper state
- Control motors

2

Terms

- Bit: can have a value of 0 or 1
- I/O pin: for input/output of data
  - Corresponds to one bit
  - 0 value corresponds to a low voltage
  - 1 value corresponds to a high voltage
- Byte: 8 bits
- Register: how we communicate with the microprocessor; each register is one byte

3

Memory Types

- Flash
  - Where the program is stored
  - Preserved when power is disconnected
- SRAM
  - Memory used when the program is running
  - "volatile" -- data stored here is lost when power is disconnected
- EEPROM
  - Memory available for non-volatile storage

4

Fuse Bits

Fuse Bits are used to customize the microprocessor. We will change the default value for four fuse bits:

- Run at 8MHz
  - Default is 1MHz
- Enable Brown-out Detection
  - Reset instead of random behavior when voltage drops
- Disable "JTAG"
  - We're using those pins for motor control
- Disable erasing EEPROM on program download
  - EEPROM can be used to store values that our program uses
  - We usually don't want to erase those values just because we have a new program!

5

Input/Output Overview

- 32 I/O pins in ATmega16
- Divided into 4 "registers": A, B, C, and D
  - Each 8-bit register corresponds to 8 input/output pins
  - Chip connections are identified with PA0, PA1, ... PA7, PB0-PB7, PC0-PC7, PD0-PD7
  - For example, the green LED is connected to PB4
- Each I/O pin is controlled separately
  - Data Direction (input or output) is specified with a DDR register
  - Outputs use a PORT register
  - Inputs use a PIN register

6

Controlling the Green LED

- Connected to PB4
- Wired so that it is:
  - ON when PB4 outputs a low voltage (0)
  - OFF when PB4 outputs a high voltage (1)
- Set PB4 as output
  - Set bit 4 in DDRB (Port B Data Direction Register)
- Value of bit 4 in PORTB controls LED
  - Clear this bit (to 0) to turn on LED
  - Set this bit (to 1) to turn off LED

7

Reading Bumpers

- Connected to PA0 and PA7
- "Touch" sends low signal (0)
  - Touch connects the two wires, connecting signal to ground
  - Input "tied high" with a "pull-up resistor," indicating 1 when no touch
- Set PA0 and PA7 as input
  - Clear bits 0 and 7 in DDRA
- Activate "pull-up resistor" for PA0 and PA7
  - Set bits 0 and 7 in PORTA
- Value of bits 0 and 7 in PINA indicate "touch"
  - 0 indicates a touch/bump
  - 1 indicates normal (no-touch) state

8

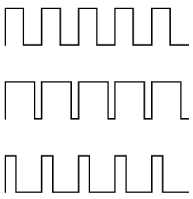
Controlling Motors

- Library functions
  - Control left and right motors separately
  - 0 is stop, 255 is full-speed forward, -255 is full-speed backward
- H-bridge chip controls motors
  - Direction
    - Left: PC3, right: PC4
    - Low (0) for forward, high (1) for reverse
  - Speed
    - Left: PD4 (OC1B), right: PD5 (OC1A)
    - Pulse Width Modulation (PWM)

9

Pulse Width Modulation

- Controls motor speed by "pulsing" power to motor
- Doesn't sacrifice torque like reducing voltage would
- Speed is proportional to the width of each pulse



10

Resources

- Atmel's documentation for the ATmega-16 microprocessor
  - Free download from www.atmel.com
- BASCOM (BASIC)
  - Installed into BASCOM-AVR-DEMO
    - Bascavr.hlp -- from Contents, see BASCOM Developing Order, Language Fundamentals, and AVR Internal Hardware
- WinAVR (C/C++)
  - Installed into doc\avr-libc
    - avr-libc-user-manual\index.html
    - avr-libc-user-manual-1.0.4.pdf

11

Programs

- See the "Sample Programs" section of the SRS Robot Level 1 Kit Manual
  - Program 1 – FlashLED, FlashLEDAlias
  - Program 2 – Whisker, WhiskerAlias
  - Program 3 – PWM
  - BumpNGo

12

Part 2

Atmel ATmega16 Programming

Part 2

Cathy Saxton
Larry Barello
Jim Wright

1

Topics

- Serial communication
  - Sending data to PC
  - Getting commands from PC
- Analog input
  - Reading light sensor values

2

Terms

- Digital
  - Two possible values, 0 and 1
  - 0 value corresponds to a low voltage
  - 1 value corresponds to a high voltage
- Analog
  - Range of values, e.g. from 0 to 255
  - Corresponds to a voltage level
  - Interpreted with an Analog to Digital converter

3

Serial Communication

- Helpful for debugging, sending commands
- ATmega16 provides support for Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART)
- Uses the “alternate functions” of PD0 and PD1
  - RxD (PD0) for receive
  - TxD (PD1) for transmit
- RS232 driver on ARC board converts between TTL (chip) voltage level and RS232 voltage level used by PC’s serial port

4

Serial Communication ~ Details

- See the Atmel doc for details on setting baud rate, data bits, parity, etc.
- Transmit and receive are enabled separately
  - Set TXEN and/or RXEN in the USART Control and Status Register B (UCSRB)
- Transmit (one character at a time)
  - Wait until the USART Data Register Empty (UDRE) signal is 1, indicating that previous communication is complete
  - Set USART I/O Data Register (UDR) to value to transmit
- Receive (one character at a time)
  - If USART Receive Complete (RXC) signal is 1, there is data in the receive buffer
  - UDR contains the next character

5

Analog Input

- ATmega16 provides an Analog to Digital Converter (ADC)
- A0-A7 have alternate functions as ADC0-ADC7 for analog inputs
- The ADC can be used to read the value (voltage level) from any ADC input and convert to a value from 0-1023
- Light sensors are connected to ADC1 (left) and ADC3 (right)

6

Analog Input ~ Details

- The Analog to Digital Converter needs to be initialized before conversions are done
  - Select the precision (8-bit or 10-bit)
  - Select the reference voltage (e.g. 2.56V, 5V)
  - See the Atmel doc for details
- Only one input can be converted at a time
  - Select channel (ADC0-7)
  - Set the ADC Start Conversion (ADSC) signal in the ADC Control and Status Register A (ADCSRA) to 1 to start the conversion
  - Wait until the ADC Interrupt Flag (ADIF) signal in ADCSRA is 1, indicating that the conversion is complete
  - Read the value from ADCH (and possibly ADCL)

7

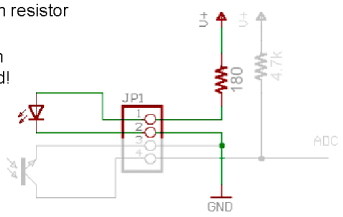
Analog Input ~ Electronics

- Fairchild QRB1134 light sensor
  - Infrared (IR) LED (emitter)
  - Phototransistor (sensor)
- QRB113x-PhotoTransistor.pdf in Datasheets folder

8

Analog Input ~ Electronics

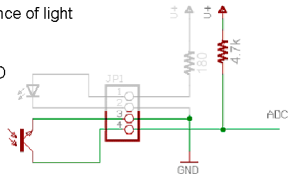
- I_F is current through LED
  - Maximum = 50 mA
  - Using a 180 Ohm resistor
  - I = V / R
    = 5 V / 180 ohm
    = 28 mA -- good!



9

Analog Input ~ Electronics

- Voltage level for ADC is determined by phototransistor and 4.7k resistor
- Phototransistor exhibits a resistance based on the amount of light it sees:
  - high resistance in darkness
  - lower resistance in presence of light
- When phototransistor exhibits a low resistance, ADC is pulled toward GND
- As phototransistor's resistance increases, so does ADC's voltage level (R of 4.7k would put ADC input at half of V+)



10

Programs

- See the "Sample Programs" section of the SRS Robot Level 1 Kit Manual
  - Program 4 – HelloWorld
  - Program 5 – InputChar, InputText
  - Program 6 – ADCPrint
  - RemoteControl

11

Part 3

Atmel ATmega16 Programming

Part 3

Cathy Saxton
Tom Saxton
Larry Barello
Jim Wright

1

Topics

- EEPROM memory access
  - Reading and writing the on-chip EEPROM
- Timer/Counter use
  - Timing activities
  - Controlling PWM
- Programming algorithms
  - Line following

2

Terms

- Interrupt
  - Action that puts normal processor operation on hold to handle an event, then returns to the main process
  - A good way to be notified of interesting events

3

Writing to EEPROM ~ Details

- Wait until EEPROM Write Enable (EEWE) signal in EEPROM Control Register (EECR) becomes 0, indicating ready to write
- Write address to EEPROM Address Register (EEAR)
- Write data value to EEPROM Data Register (EEDR)
- Set EEPROM Master Write Enable (EEMWE) signal in EECR
- Set EEPROM Write Enable (EEWE) signal in EECR
- (Some timing issues, see Atmel documentation for details)

4

Reading from EEPROM ~ Details

- Wait until EEPROM Write Enable (EEWE) signal in EEPROM Control Register (EECR) becomes 0 (can't read while write is in progress)
- Write address to EEPROM Address Register (EEAR)
- Set EEPROM Read Enable (EERE) signal in EECR
- Read data value from EEPROM Data Register (EEDR)

5

Using a Timer/Counter for Timing

- Timer/Counter 0 is convenient as a timer
  - Value changes at a specific rate
- Delay loop isn't as good
  - Inaccurate because it can't compensate for interrupts that take processor time

6

Timer/Counter ~ Details

- Timer/Counter 0 setup
  - See the Atmel doc for details on making these settings
  - Clock source determines counting speed
    - A prescaler that divides the controller's 8 MHz by 1024 will result in a counter running at 8 kHz
  - "CTC" (Clear Timer on Compare Match) waveform generation mode
    - OCR0 specifies "match" value, which resets counter to 0
    - An OCR0 value of 8 causes a match every 1 millisecond
  - Enable interrupt for notification of compare match
    - Set Timer/Counter 0 Output Compare Match Interrupt Enable (OCIE0) in Timer/Counter Interrupt Mask Register (TIMSK)
    - Set the I-bit (Global Interrupt Enable) in the Status Register (SREG) to enable interrupts

7

Timer/Counter ~ Details

- Using Timer/Counter 0
  - Increment variable when get compare match interrupt
    - This counts milliseconds
  - Wait function can use count for accurate timing
  - Remember current value, then compare later for elapsed time
    - Lets other processing occur simultaneously with timing
    - Measure time required to perform a task
    - Watch for time limit, e.g. for end of line maze run, or to change behavior when looking for a sumo robot

8

Motor Control Using an H-bridge

- Motor control is done through an H-bridge
- The H-bridge receives low-power inputs from the microcontroller
  - Direction and speed (PWM) for each motor
- The H-bridge sends high-power outputs to drive the motors
  - Each input to the H-bridge controls one connection to the motor
  - A high input will connect the motor to power
  - A low input will connect the motor to ground

9

Pulse Width Modulation (PWM)

- Motor motion is determined by voltage applied at inputs
  - Positive on one input and ground on the other will turn the motor in one direction
  - Swapping power and ground will turn the motor in the other direction
  - Applying the same voltage (either power or ground) to both inputs will cause the motor to stop
- Using PWM sends power and ground for the duration of the pulse width, then sets both inputs to the same level until the next pulse starts

10

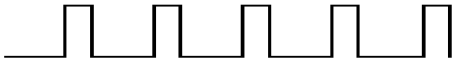
Pulse Width Modulation (PWM)

- Moving forward
  - We send a low (0) value for the direction signal; that results in one input of the motor being connected to ground
  - The PWM signal controls the other motor input
    - When the PWM signal is high, the motor turns
    - When the PWM signal is low, the motor stops
  - So, the PWM signal needs to be high for the width specified, then low until the next pulse starts; the image below shows an example that will run the motor at 80% of full speed

11

Pulse Width Modulation (PWM)

- Moving backward
  - We send a high (1) value for the direction signal; that results in one input of the motor being connected to power
  - The PWM signal controls the other motor input
    - When the PWM signal is low, the motor turns
    - When the PWM signal is high, the motor stops
  - So, the PWM signal needs to be *low* for the width specified, then *high* until the next pulse starts; the image below shows an example that will run the motor backward at 70% of full speed



12

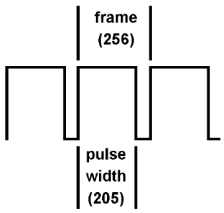
Using a Timer/Counter for PWM

- Timer/Counter 1 can be used to automatically create the appropriate waveforms on two "Output Compare" pins
  - These outputs are connected to inputs of the H-bridge on the ARC board
  - OC1A (alternate function of PD5) controls the speed for the right motor
  - OC1B (alternate function of PD4) controls the speed for the left motor

13

Timer/Counter ~ Details

- Count from 0-255 using "Fast PWM" waveform generation mode (frame width = 256 units)
- Example: Run the right motor forward at 80% of full speed
  - Right motor is connected to OC1A
  - Set Output Compare Register OCR1A to 205 (80% of 256)
  - Set the "Compare Output Mode" to set OC1A at the beginning of the frame (when the counter wraps from 255 to 0) and clear OC1A when the counter matches OCR1A



14

Line Following Algorithms

- Simple Version
  - Watch for losing the line on either side
  - Turn to recover line
  - Results in noticeable "wiggle"

15

Line Following Algorithms

- Better Version
  - Read Larry Barello's paper
    - http://www.barello.net/Papers/LineFollowing/index.htm
  - Try to keep sensor readings balanced
  - Steering correction proportional to difference in sensor readings
  - Once it's tuned, no more wiggling
  - Only one state!

16

Programs

- See the "Sample Programs" section of the SRS Robot Level 1 Kit Manual
  - Program 7 – EEPROM
  - Program 8 – Timer
  - Program 9 – LineFollow

17

Part 4

Atmel ATmega16 Programming

Part 4

Cathy Saxton
Tom Saxton
Larry Barello
Jim Wright

1

State Machines

- Divide complex task into small pieces
- Each piece quickly handles a case or state
- Control moves from one state to another
- Logic can be event driven
- Algorithm can be one piece of a program without knowledge of other pieces

2

State Machine Advantages

- It's easier to read and maintain small, well-defined pieces than one huge spaghetti pile
- Fast and interruptible
- Works well with other simultaneous processes, like data collection

3

State Machine Examples

- Sumo robot logic
- Mechanics of following lines in a maze
- Parsing incoming serial data, possibly inside interrupt handler
- Data driven algorithms, like dead reckoning

4

Sumo Robot States

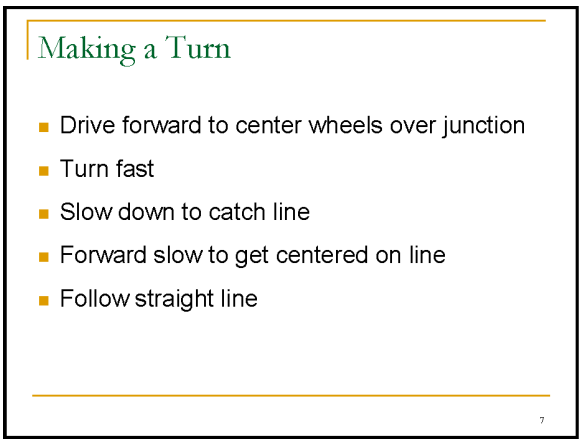
- Start maneuver
- Rotate and scan
- Have target robot in sight
- Have target robot centered
- Approaching target
- Push to the white ring
- Back away from white ring

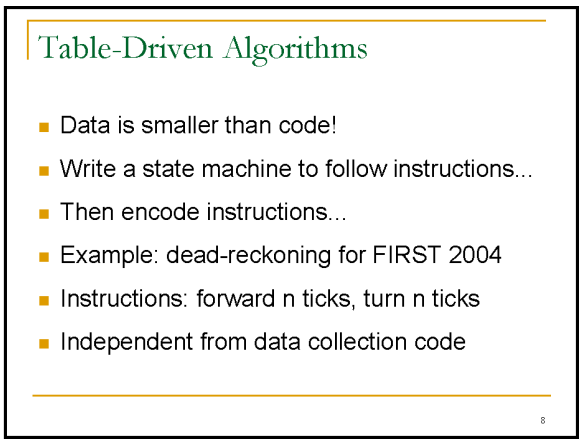
5

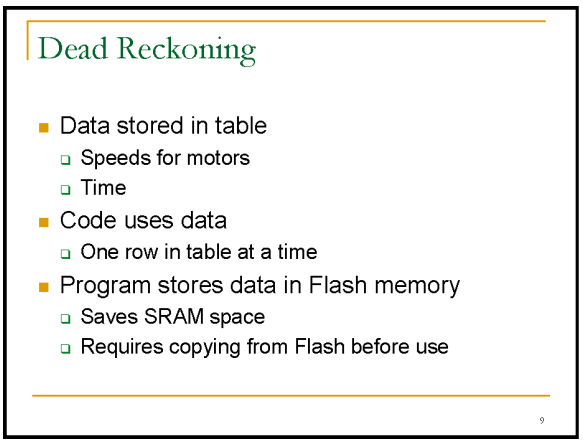
Junction Detection

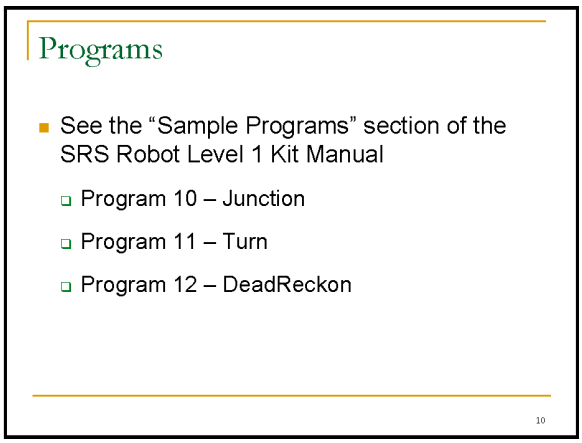
- Follow straight line
- Scan junction
- Pass result to maze navigation logic...

6









Sample Programs

There are sample programs available from http://www.seattlerobotics.org/WorkshopRobot/Level1/ (or the Workshop CD's "Samples" folder). There are versions for BASCOM (BASIC) and C. The program files contain descriptive comments to explain what the code is doing.

The programming concepts are explained in the slides in the "ATmega16 Programming" section. The "Program Descriptions" section below describes the programs that are listed at the end of each section of slides. The *extensions* are suggestions for modifications you can make to the code to experiment and learn. The *challenges* are optional, harder modifications.

Please see the "AVR Robot Controller 1.1 Software Tools" section for instructions on installing and using the software to program the microcontroller.

Please see the "AVR Robot Controller 1.1 Hardware Description" section for details on the I/O pins used by the SRS Robot.

Creating New Programs

For some of the challenges, you may wish to create a new program instead of modifying one of the sample programs.

A good way to get started with a new program is to make a copy of a folder for an existing program, then rename the new folder and its .bas or .c file. (For C programs, you will also need to edit the makefile to reflect the name change for the .c file. Look for the SRC= line. You may also change the PRG= line for consistency.)

C Programming

Each sample program is in its own folder. Each of these folders contains the C files needed for the program and a "makefile" that provides information for the tools. All the files created by the build process are stored in this folder as well. The concept of a makefile will be new to many people, even experienced programmers. You do not need to understand the makefile, but it's important that you do not rename it!

The makefile is set up to use the BASCOM programmer (supplied with the kit). If you will be using an AVRISP, you'll need to override the settings in the makefile. You can modify the makefile (in each program folder), but a better way is to define environment variables:

AVRDUDE_PROGRAMMER = avrisp
AVRDUDE_PORT = com1 (*see the Software Tools section for Mac information*)

When using Programmers Notepad, but sure to use LaunchPN.bat (from the "C Samples" folder) to start it. To build a program, open a C file that is part of the program you want to build. Make any edits you want, save the file, then choose the Tools/Make All command to build. This will automatically use the makefile in the same folder as your C file. An "output" window will show the results of your build. (If you get a message that "the system cannot find the file specified," you probably just need to restart Programmers Notepad using LaunchPN.bat.) To download the program, make sure the robot is on, then choose the Tools/Program command.

(You can also use the command line to build and download your programs. Run setpath.bat (from "C Samples") in the command window to set up the environment, then call make manually.)

Program Descriptions

Program 1

Purpose: Learn how to use outputs to make the green LED flash.
Programs: FlashLED, FlashLEDAlias -- note how the code in FlashLEDAlias is easier to read; you don't need the comments to explain what's going on.
Extensions:

- Change the rate at which the LED flashes.
- Change the code so that the LED is on longer than it is off.

Program 2

Purpose: Learn how to use inputs to detect obstacles, lighting the LED when a bumper (whisker) is touched.
Programs: Whisker, WhiskerAlias
Extension: Add code so that touching either bumper turns on the LED.
Challenge: Make the LED flash at four different rates when bumpers are: open-open, open-closed, closed-closed, closed-open.

Program 3

Purpose: Drive! Learn how to control the motors.
Program: PWM
Extensions:

- Experiment with other PWM values, e.g. set both motors to 255 or -128.
- What happens when you use 128 for the left motor PWM and 255 for the right motor PWM?
- What happens when you use -100 for the left motor PWM and 100 for the right motor PWM?
- Flash the LED once a second for five (or three) seconds before starting.
- Wait for a bumper touch (instead of delay) before starting the motors.
- Make a bumper touch cause the robot to back up, then stop.
- Make the robot go forward 10 inches, then turn left 90 degrees.
- Make the robot do the previous extension 4 times.

Challenges:
- Make a bumper touch cause the robot to back up, *turn away from the obstacle*, and then resume exploring. (The BumpNGo program shows one possible solution, including waiting for a bump at the beginning of the program before starting.)
- Teach the robot how to navigate a (specific) obstacle course.
- "Bounce" along a wall: drive until you touch a wall, then turn away from it and arc back toward it, continuing to "bounce" along.

BumpNGo

This program is an example of a solution to the first challenge in Program 3. It waits for a bump at the beginning of the program, then drives forward until it bumps something, at which point it backs up, turns away from the obstacle, and then continues forward again (continuing to avoid obstacles).

Program 4

Purpose: Learn how to send text from the robot to a PC via a serial connection. The text is received by a program such as HyperTerminal on Windows.
Program: HelloWorld
Extensions:
- Change the text.
- Send the text once a second.
- Print "open" or "closed" based on the state of the left bumper.

Challenges:
- Print text once a second until either bumper is closed.
- Print "open" or "closed" based on the state of the left bumper, but only print it when the state changes. (Hint: you will need a variable to remember the current state.)
- Print open and closed messages (as in the previous challenge) for both bumpers.

Program 5

Purpose: Learn how to receive text/commands to the robot from a PC via a serial connection.
Programs: InputChar, InputText
Extension: Change InputText to print a special message when the program receives a '?'.
Challenges:
- Use serial input to control the robot, e.g. moving forward, turning left/right, and moving backward in response to 'f', 'l', 'r', and 'b'. Have each motion performed for 1 second, then wait for more input.
- Take code from a couple of your previous programs, create functions for those activities, and use input to determine which activity to perform. Have your program start by sending text with a menu of options.

Program 6

Purpose: Learn how to read analog input values from the light sensors. This program sends the values to a PC via a serial connection.
Program: ADCPrint
Extension: Add output for the second sensor.
Challenge: Drive forward until the light sensor reads much higher or lower than the initial reading

RemoteControl

This program uses input from the numeric keypad to drive the robot in short bursts. It prints (via serial output) the values of the light sensors each time a character is received (any character works, not just "driving" keys). This program combines driving, serial input, serial output, and reading of analog sensors.

Program 7

Purpose: Learn how to read from and write to the EEPROM memory. This program reads and stores the light sensor values.
Program: EEPROM
Extension: Show sensor values while you are waiting for a key press (e.g. every 200 ms).

Program 8

Purpose: Learn to use the Timer/Counter for timing. This program flashes a LED at a specific rate while reading sensors.
Program: Timer
Extensions:
- Change the flash rate.
- Add more code to the loop, e.g. showing sensor values. Notice that the flashing still maintains its rate, with no changes necessary.

Challenges:
- Modify your EEPROM program to use the timer instead of WaitMs when showing sensor readings.
- Read sensors for 10 milliseconds and calculate the average value.

Program 9

Purpose: Learn how to use proportional steering to follow a line.
Program: LineFollow
Extension: Tune values for your robot. Adjust the light thresholds (use ADCPrint to see sensor values). Adjust the scaling proportion for steering.
Challenge: Go faster without falling off the line.

Program 10

Purpose: This program shows how to implement the states involved in detecting and identifying a junction. It will determine which options are available (left, right, forward), then stop.
Program: Junction
Extension: Tune values for your robot.
Challenges:
- Detect end of maze (solid 6” diameter black circle).
- Use sensor sample averaging -- take several readings and use their average when making decisions.
- Save light sensor thresholds in EEPROM.

Program 11

Purpose: This program shows how to implement the states involved in turning at a junction.
Program: Turn
Extension: Tune values for your robot.
Challenge: Go faster!

Program 12

Purpose: This program shows how to use a table to store a sequence of control commands for the robot. It also demonstrates how to store constant data (the table) in program memory (Flash) instead of the default SRAM.
Program: DeadReckon
Extension: Modify/Add commands to the table.
Challenge: Add another piece of data to the state, e.g. LED on/off.

AVR Robot Controller 1.1 Hardware Description

The Level 1 Robot Kit ships with an ATmega16 microcontroller. The Atmel AVR microcontrollers are designed to be pin- and code-compatible where possible. In this case, for example, you could substitute an ATmega8535 or ATmega32. These chips will all function in the ARC board with minimal software changes (typically just a re-compile), the primary difference being the amount of internal memory available. Please see Atmel's site (www.atmel.com) for a list and descriptions of available devices.

Power

Power can be any source from 6-24 volts DC. Pay attention to the polarity symbols near the power plug -- ground is in the center, with +V applied at either end. The ARC board is protected against battery reversal with a diode. This diode reduces the available power to the board and prevents destruction of the H-Bridge in case of accidental battery reversal. If you want, you can use a polarized battery connector and solder it directly to the board and replace the diode with a bare wire.

I/O Pins

The ATmega16 microcontroller has four 8-bit I/O ports and many peripheral functions that work through these I/O pins. Port A also doubles as the analog input port. Port D doubles for many of the built in functions like serial I/O, PWM and timers. Port B doubles as the programming interface.

The following table lists the I/O pins that are brought out to headers on the ARC board. (The expansion headers JP3 and JP10 are not included since they bring out all I/O port bits as well as power and ground.) *Connection* lists what device or feature the robot communicates with. *Physical Pin* is the connection to the ATmega16. *Functional Name* indicates how each connection is used in the programs. *Input/Output* describes the communication direction as viewed from the ATmega16. *Header* is the text identifying that connection on the board. (See the schematic for pin details.)

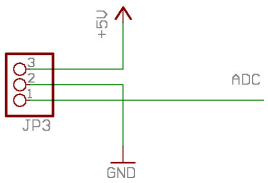
Connection	Physical Pin	Functional Name	Input/ Output	Header
Left bump sensor	PA0	PA0	input	LEFT (at front of board)
Right bump sensor	PA7	PA7	input	LEFT (at front of board)
Left distance sensor (Level 2)	PA0	ADC0	input	RIGHT (at front of board)
Right distance sensor (Level 2)	PA7	ADC7	input	RIGHT (at front of board)
Left light sensor	PA1	ADC1	input	LF ("left floor")
Center light sensor (not included with kit)	PA2	ADC2	input	CF ("center floor")
Right light sensor	PA3	ADC3	input	RF ("right floor")
Left motor direction	PC3	PC3	output	LEFT (at back of board)
Left motor PWM	PD4	OC1B	output	LEFT (at back of board)
Left motor channel A (encoder, Level 3)	PD3	INT1	input	LEFT (at back of board)
Left motor channel B (encoder, Level 3)	PC2	PC2	input	LEFT (at back of board)
Left servo (not included with kit)	PC2	PC2	output	LEFT (at back of board)
Right motor direction	PC4	PC4	output	RIGHT (at back of board)
Right motor PWM	PD5	OC1A	output	RIGHT (at back of board)
Right motor channel A (encoder, Level 3)	PD2	INT0	input	RIGHT (at back of board)
Right motor channel B (encoder, Level 3)	PC5	PC5	input	RIGHT (at back of board)
Right servo (not included with kit)	PC5	PC5	output	RIGHT (at back of board)
Serial input	PD0	RXD	input	JP5
Serial output	PD1	TXD	output	JP5
Green "program" LED	PB4	PB4	output	PROG (green LED)
Option A	PB6	PB6		ISP/JP9, A
Option B	PB7	PB7		ISP/JP9, B
Option C	PB4	PB4		ISP/JP9, C

Expansion Headers JP3 and JP10

Refer to the schematic for the connections to these headers. All CPU I/O, +5v, ground, +/- 10v and battery voltages are supplied. JP3 supplies +5v and ground from the analog section. JP10 supplies +5v, ground, +10v and –10v from the digital section. The +/-10v is from the serial interface chip and can only supply about 10ma of current. This is sufficient for one or two low powered op-amps. *JP3 and JP10 are oriented with pin 1 toward the notch side of the CPU.*

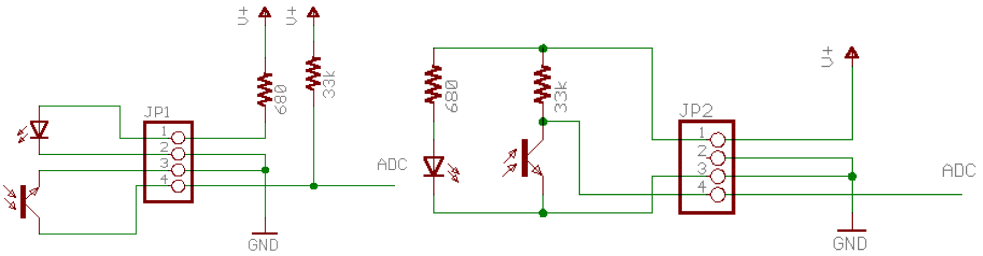
Left and Right Analog Inputs

These are the headers labeled “LEFT” and “RIGHT” at the front of the board, with the 3-connector sockets. They are raw analog inputs with filtered +5v and ground. They are intended to connect directly to the Sharp GP2D12 or 15 sensors. However any 0-5v output sensor may be used. You can also connect a potentiometer to these inputs for position feedback or variable input.



Left, Center, and Right Floor Sensor Inputs

These are the headers labeled “LF,” “CF,” and “RF” at the front of the board, with the 4-connector sockets. These match the pin-out of QRB1134 phototransistors. The resistors supplied with the kit are selected to work well with these sensors. Pin 1 is attached to the outer E (emitter) wire. Pin 4 is attached to the outer S (sensor) wire. If you do not want to commit the board to these sensors, you can replace the 180-ohm resistor with a piece of wire and leave out the 4.7k resistors. The resulting circuit is similar to the Left and Right analog inputs and allows +5v-powered sensors to be connected to the board.



Normal Wiring Alternate Wiring

Left and Right Motor Connectors

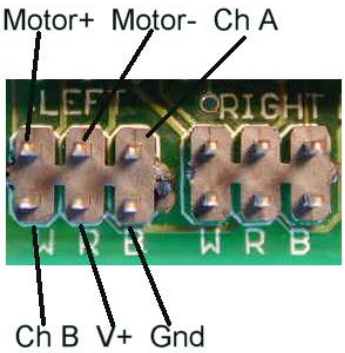
These are the headers labeled “LEFT” and “RIGHT” at the back of the board, with the 2x3 pin headers. The left and right motor connectors provide output from the H-bridge, power, ground, and two CPU I/O lines. The additional lines can be used to implement an encoder feedback or input mechanism, or can be used as output lines to drive a standard R/C servo.

H-bridge output

The H-bridge output is on the inner pins of the header and is directly connected to the 754410 H-bridge. This can supply up to 1A with a battery voltage of up to 24V. If you gut a servo and re-attach the servo cable wires to the motor, be aware that the two H-bridge output pins are not aligned with the power pins: you either need to wire the servo motor to Red, White, or connect the plug in reverse.

Encoder input

When driving motors with the H-bridge, the additional two signal wires and power can be used to gather encoder feedback. When driving an encoder make sure the jumper block labeled **SERVO** is connected for +5v.



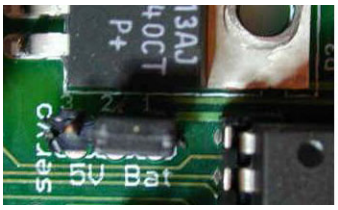
	Left Motor	Right Motor
DIR	Portc.3	Portc.4
PWM	Portd.4 (OC1B)	Portd.5 (OC1A)
CH-A	Pind.3 (INT1)	Pind.2 (INT0)
CH-B	Pinc.2	Pinc.5

R/C servo output

When driving an R/C servo, use the pins nearest to the edge of the board. They are marked with letters indicating polarity: “W R B” for White, Red, and Black. Some servo brands substitute yellow for white.

When driving servos, the board has the option of selecting regulated +5v or raw battery voltage to drive the servos. R/C servos are usually rated at 4.8v and 6.0v, but can be driven with 9-12 volts without damaging the internal electronics. Higher voltages will increase speed and torque of the motors. A rough rule of thumb is that doubling the voltage (9.6 vs. 4.8) doubles the RPM and the maximum torque of the servo.

Power Source



A 3-pin header is used to select the power source for the Motor Headers (V+ in the picture above) when used for R/C servos or encoders. There are two options: regulated +5 volts, or the battery supply. Use the shorting jumper to make this connection. The H-Bridge is always connected to the main power source.

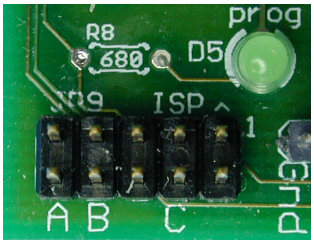
In-System Programming and Option Select

The 2x5 pin header labeled ISP (and JP9) is for In-System Programming of the CPU. This is the mechanism by which the CPU is loaded with a new program. This connector works with any Kanda-compatible 10-pin AVR programmer. The cable connects to the ISP header with the pin 1 marker (a triangle) next to the 1 and caret printed on the board.

The header has a dual use. When the programming cable is removed, several of the pins may be used to select options or give feedback. All pins are available from the expansion headers – refer to the schematic for more details. In addition, one pin is used to control the program LED (green). This gives both visual feedback while programming, and can be controlled by the user’s program to give feedback.

When a shorting bar is across the inputs, the corresponding pin value will be zero. There are no pull-up resistors on these inputs, so you need to enable the CPU pull-up on those ports if you intend to use them as inputs.
Reset has no CPU readable input. It simply resets the CPU when shorted.

Pin	Pair label	Direction	IO port
9	A	Input	Pinb.6 (MISO)
7	B	Input	Pinb.7 (SCK)
5		Input	CPU reset
3	C (Prog LED)	Input/Output	Pinb.4/Portb.4
1		Output	Portb.5 (MOSI)



Options I/O port assignments

Program LED output

The program LED output is inverted. I.e. writing a 0 to portb.4 will illuminate the LED, writing a 1 will turn it off. Remember to configure that I/O pin as an output or it will never turn on. When using Option C as an input, you cannot use the LED (they are connected together).

AVR Robot Controller 1.1 Schematic