

Building a Line Following Robot

As a programming teacher, I frequently adopt the attitude of "Come inside a programmer's brain!" I can then explain the concept or source code as I would if I did all my thinking out loud. This is helpful to many students because most of my examples do come from my brain or experience. This allows me walk them through the process and to make mistakes or false assumptions, and then correct them as we proceed.

I would like to take that attitude here as I describe the processes I am going through in developing a line-following robot.

This all started recently when I was told I would soon be teaching a class called "Embedded Systems" at my college. When I learned that the chip was a PIC16F series microcontroller, and that there would be a development board, similar to the Handy Board, I realized I had some experience in this area. For my Bachelor's Degree in Electronics Engineering, I had been the programmer for an autonomous robot that used a Motorola 68HC11, a similar chip.

It occurred to me that the "Final Exam" for this class could very well be a line following robot. The project is simple enough to be "do-able" in college electronics and programming students, complex enough that it would be a challenge and learning experience for students. Line followers are popular enough in robotics clubs, that students could find plenty of supporting material on the Internet. I Googled "line following robot" and found many good leads and great sites describing what is going on with line following robots.

Next, I need a "Mission Statement" for what I want to do. Later, this can be revised as I learn more specifics.

The Mission: Develop an autonomous robot that will follow a black line on a white background.

I picked black on white because it seemed the most often used course.

Some Restrictions: Since I know the chip I'll be teaching, the chip PIC18F8680 had to be the mandatory microcontroller. The class also does one week of assembly language introduction, then shifts to C, so C will be the language. Other than that, I want to be open to discovering many possibilities.

WHAT'S OUT THERE?

The first task appeared to be taking a look at what types of line following contests were out there. It made sense to me that the ultimate test of any robot is to have it compete against other robots. I searched and looked at many articles, photos and videos. Here's what I saw.

The courses are either black lines (usually $\frac{3}{4}$ inch electrical tape) on white background, by far the most common, or white lines on black background. Since it was a simple matter to adjust the robot behavior for either, I elected to work with black lines.

Next, I saw that there seemed to be two types of courses. Those where the race course was composed of a fixed number of tiles with specific configurations, like Figure 1:

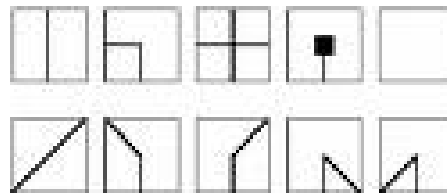


Figure 1: Line Following Race Course Tile Set

Then with a box full of these tiles, a race course of any desired shape could be constructed by just selecting the desired combination of tiles.

The remainder used a course either pre-drawn on a playing surface, or laid down with electrical tape on the race course like this:

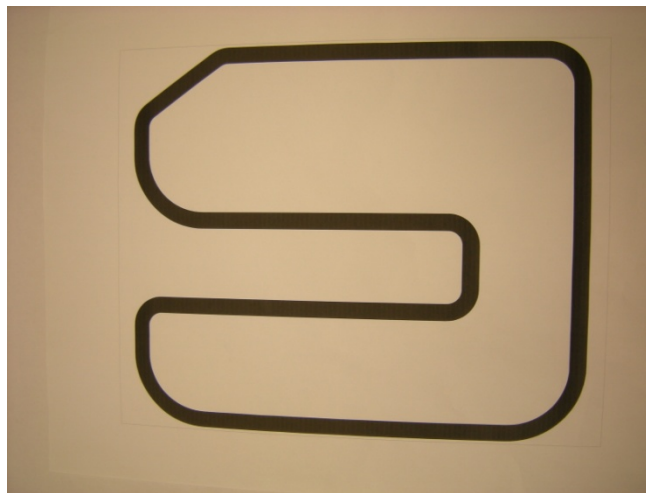


Figure 2: “Easy” Line Following Course, 3 X 3 feet, Made by CAD Students

I noticed that the contest courses came in three different general difficulty levels:

1. Easy: Where the track has gentle, usually 6-inch radius curves.
2. Medium: Where the lines either cross each other and/or there are up to 90 degree sharp turns, or
3. Hard: With crossings and often greater than 90 degree turns.

And of course, all of the contests put a premium on speed by awarding first prize to the robot that finishes in the fastest time.

NARROWING THE CRITERIA

With what had observed, and what I know, the following criteria begin to emerge:

The robot has to be fast. In the videos on the net and on YouTube.com I observed some pretty fast robots!

The robot has to properly navigate difficult courses. It didn't make sense to design a robot for an “easy” course, only to find later that I had to discard the entire project because it wasn't able to do a more demanding task. Mission Statement, Revision 1:

The Mission: Develop a fast, autonomous robot that will follow a complex black line course on a white background.

THE PROPULSION SYSTEM

I was amazed at the variety of methods that participants chose to get their robots moving.

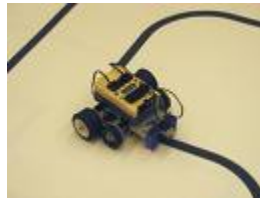
Tank treads: [1]



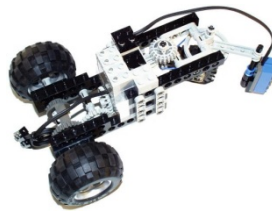
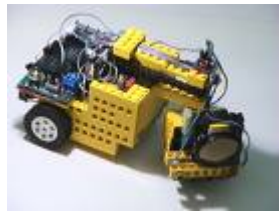
Six Wheels: [2]



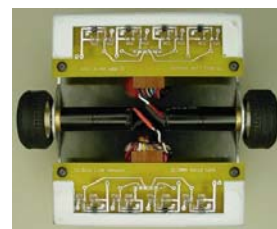
Four Wheels: [3], [4], [5]



Three Wheels: [6], [7], [8]



Two Wheels: [9], [10], [11]



And even something that looked like a robot in a bottle: [12]



WHICH CONFIGURATION?????

As with any other complex project, this one will present a number of opportunities in which to make decisions. The two-wheeled, or two wheels with caster seems to be the favorite, but I need to see if I can apply some of my knowledge to this situation.

I've heard enough about problems with tank treads. Therefore I could easily eliminate those. I can't imagine that six-wheelers are any better, and anything a four wheeler can do, a tricycle or two-wheeler with a caster can do better. Therefore, four to six wheels didn't seem to be the way to go.

I was intrigued by an arrangement I saw on the net that I'll call a tricycle. The sensors and a turning wheel were way out in front, and the bulk of the robot weight was over the two drive wheels in the rear. The speed seemed phenomenal! The advantage seems to be a much faster response time by getting the turning wheel up front and near the sensors.

My other favorite vehicle is plain vanilla two side wheels with independent drives and a free-spinning swivel wheel or caster. I've seen these robots function and I really like the responsiveness and simplicity of just two wheels.

I spent several days thinking and drawing sketches and trying to visualize both configurations. I saw no way to combine the best of both, and kept returning to two choices. The tie breaker came when I looked at the task from the eyes of my future students. Which one would be best for them? Based on the ease of the concept of two wheel drive and the difficulty of front wheel servo steering and motor control, it seemed the two-wheeler was the better choice for a beginner. I also know that I often have students who would pick the tougher project for a more rewarding learning experience. So I divided the project into two phases:

Phase 1: Two independently powered wheels with swivel wheel or caster.

Phase 2: Tricycle arrangement with sensors and servo controlled wheel up front and a "trailer" for electronics, motors, wheels and battery.

I will give students a choice or have them do Phase 1 first and Phase 2 later, or let each team select the desired configuration.

Personally, I decided to do Phase 1 now, and then use what I learn to proceed on to Phase 2. So, I will do both, but for now, just the two-wheeler.

THE BLOCK DIAGRAM

Once the main configuration is chosen, the first thing to do seems to be to make a functional block diagram. Although it might grow or change later, I always like to have

“the big picture” available. Since I teach Microsoft Visio 2007 in one of my classes, I decide to make a Block Diagram in Visio.

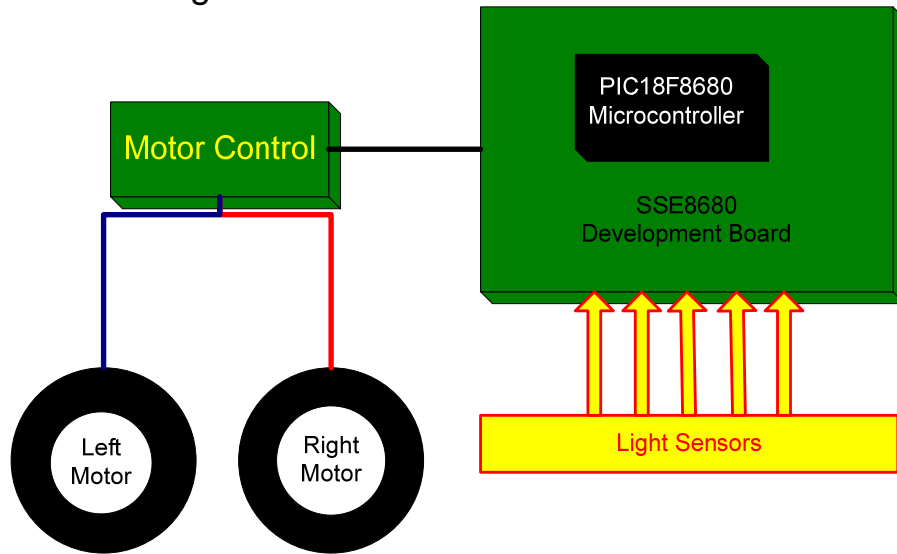


Figure 3: Line Following Robot Block Diagram.

Just looking at this drawing really gets brings up a host of ideas and questions. Hardware, software, logic, motor control and many other thoughts come rushing in. I need to slow down and compartmentalize.

Since this project is the basis of beginner’s robotics projects or contests in just about every robotics club on the planet, the drawing provides an outline for how to structure a class to teach the different aspects of robotics. The robot’s program will require the four main ingredients of every computer, microprocessor, microcontroller and the programs that run in them. Here are the four and how they relate to line following project:

Input: Read the white/black on the floor and condition the input signal(s) for transmission into the "brain" or computer/MPU/CPU in a way that questions can be asked and decisions made.

Process: Based on the inputs received, decide what change (if any) needs to be made to the robots speed and direction. Convert the results of any decisions made into something that can be sent to motor speed control and/or steering.

Output: Send the old or the newly adjusted control signals to speed and/or steering devices.

Storage: The first thing needed is a place to store the computer program to do the "process" above. We will need to store sensor readings, speed or direction information or a number of other things, but they will become clearer as the project progresses.

QUESTIONS.?.?.?

The following immediate questions become apparent:

Input:

- What type of line detection sensors? White light, LEDs, infrared, or ???
- How many sensors?
- What sensor configuration? Straight line? Vee? Inverted Vee?
- How do light readings get relayed to the MPU in a useful form?
- Digital, analog?
- ...And how do we get input information quickly?

Process:

- What processor to use? (PIC18F8680 Predetermined)
- What programming language? (C with a little assembler predetermined)
- What line following algorithm to use??? (Depends on the sensor arrangement.)

Output:

- DC motors, servos or stepper motors?
- Voltage and power (battery) requirements??
- Will there be one power supply or two?
- What about motor speed control issues?
- Steering!?! Two wheels and a caster – yes, but motor control?
- PID Steering control or not?

Other:

- Power supply or supplies, when determined will have a big effect on mass.
- Batteries: Probably the heaviest part of the robot. How to factor this in?

INPUT SENSOR SELECTION

After many trips to Google, I started narrowing this choice. I was not impressed by "big" solutions where large, unfocused lights or LEDs flooded the floor with light. I saw several of these solutions that required additional shielding and caused problems because the total light "under the hood" was too intense.

I was drawn to three different sensors:

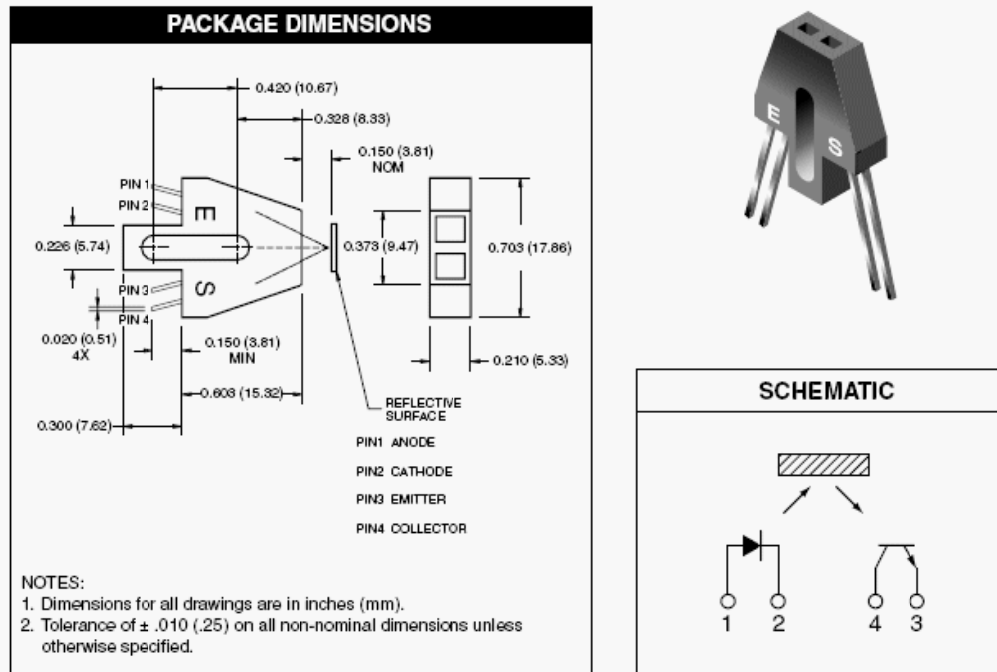


Figure 4: QRB1114 Optical Sensor

I have used the QRB1114 shown in Figure 4 before in a previous project. Notice the oval hole in the center. I like this because a small nut and bolt can be placed through this opening and make the sensor somewhat adjustable. So this is a definite candidate.



Figure 5: R185-Single-Line-IR Sensor

Figure 5 shows the R185-SINGLE-LINE-IR from Lynxmotion. It is attractive because it has on board electronics and a status LED to assist in troubleshooting. Notice it also has the oval hole to facilitate mounting adjustments.

I was put off by the \$14.95 cost, because I believed I can do everything this sensor does for under \$2.00. I also teach “starving students” and don’t want to sticker shock them with this sensor. If they choose to use 4, 6 or 8 in-line sensors, this could get expensive. So for me personally, this would be the choice, but for a robot that I can show off to my students as “affordable”, I choose to be on a tighter budget.

Then I stumbled across this:



Figure 6: QRD1114 Optical Sensor

Although the Figure 6 photo makes the QRD1114 look big, it is really tiny! Here is a link to the complete [QRD1114 Data Sheet](#), but I’d like to point out some particulars.

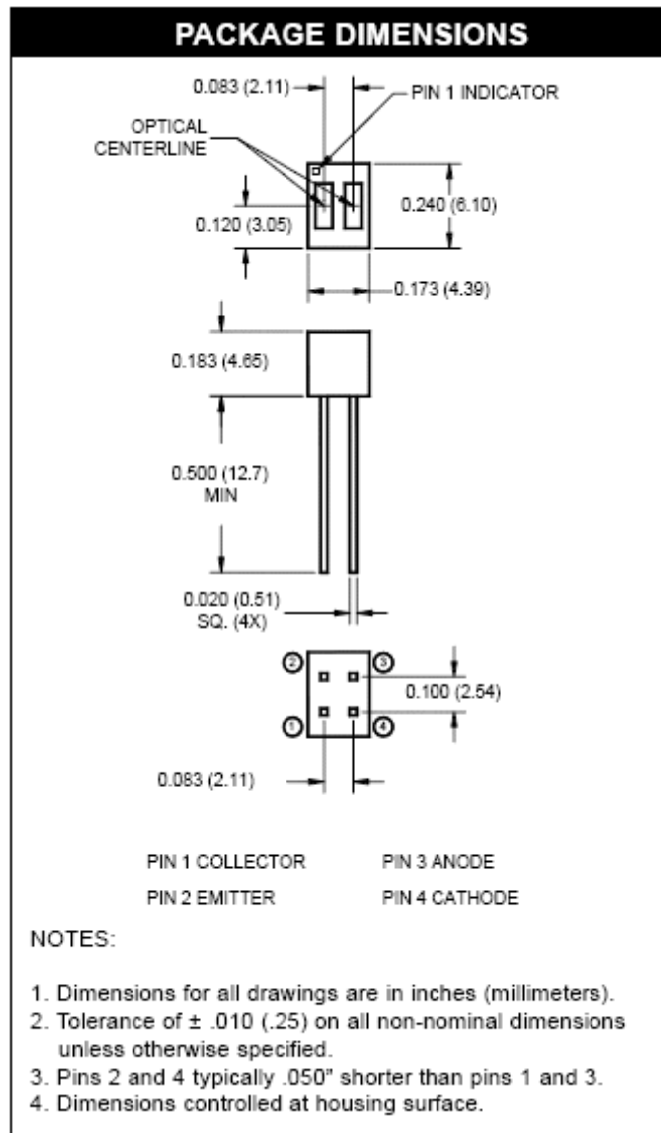


Figure 7: QRD1114 Package Dimensions

At 6.1 X 4.39 millimeters, this was the smallest I'd seen. And the price was excellent. The surface mount version of this sensor was also used in a very fast robot I saw in RobotRoom (See far right photo in the line of four two-wheelers above. [11])

SENSOR PLACEMENT

The simplest form of line follower uses one sensor. Let's say that the course is a black line on white. The robot, if programmed for "Left" is placed with the sensor just to the right of the line over the white surface. Over white, it is told to go left until it sees black. Then it is told to go right until it loses the black line. So... this is not really a line follower, but an edge follower. The main drawback for the single sensor seems to be

speed. The robot spends all its time turning or bouncing off the edge of the line. I could see no way to get more speed without more accurate input information.

Once you leave the one sensor idea behind, you naturally go to two sensors as better. But wait! If 2 are better, how about 4, or 6, or 8, or some really big number?

Here is what two sensors looks like:

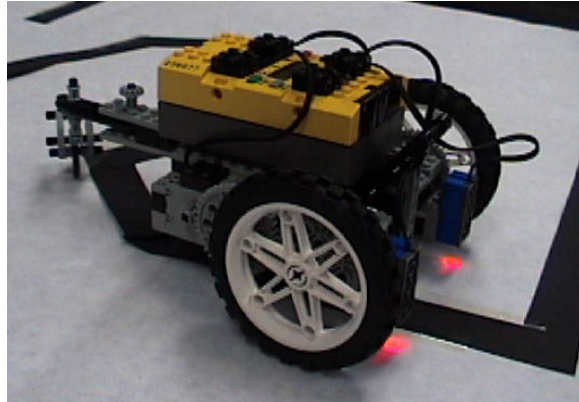


Figure 8: Two Line Sensors

And 4 sensors:

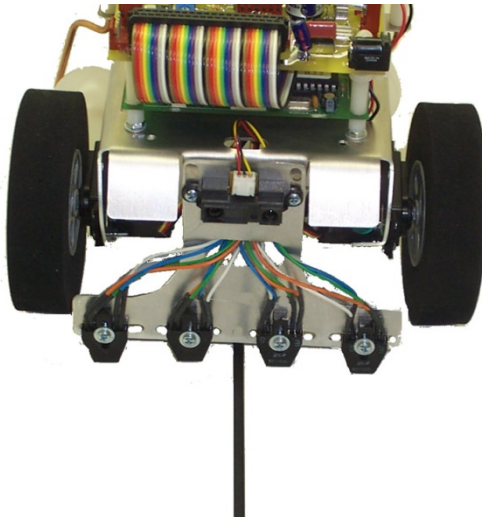


Figure 9: Four Line Sensors

And 6 sensors: [13]

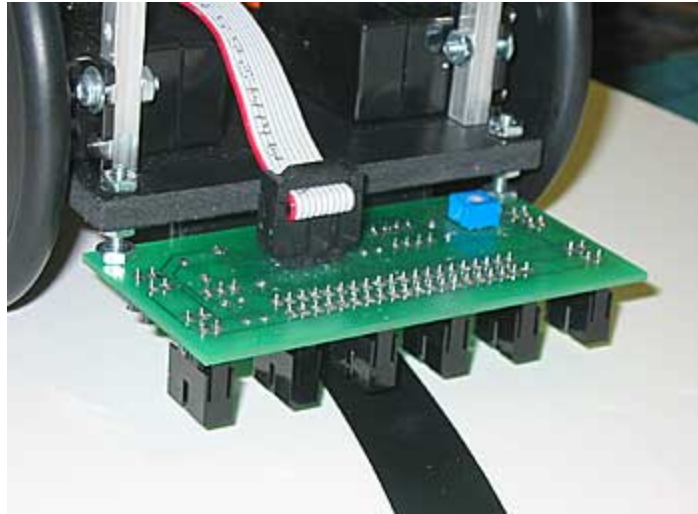


Figure 10: Six Line Sensors

Look at the 6 sensor robot above. You can see that the line is under the 3rd sensor, so the robot is a little to the left of where it wants to be to stay centered. If the black line was under the sensor to our left in the photo above, we would know that the robot is much farther from center, and we need a more dramatic steering correction. It seems that more sensors can provide more information **IF WE CAN USE IT PROPERLY AND REACT QUICKLY.**

Just dialing in a sharper turn is not the answer here because the robot would tend to overshoot and overcorrect, making the steering erratic and unstable. Fortunately, there are several fixes for this. The most common is called PID. Proportional, Integral, Derivative are three mathematical feedback terms that can be used to reduce overshoot and hunting. I won't cover that here. There are plenty of web articles on the subject. A second solution would be to write a program smart enough to know the difference between:

- Off a little to the left (from a perfect center position), and
- Off a little to the left (coming back from off a LOT to the left!)

The first requires making a course correction to the right. The second requires you start straightening out from your previous stronger corrective turn to the right.

Suffice to say that four or more sensors, with PID or a "smart" algorithm, can get you more speed than the one or two sensor robot. I was doing great with all this until I saw these two photos:

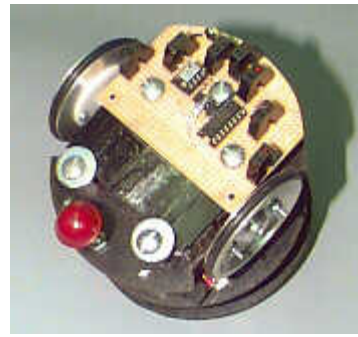
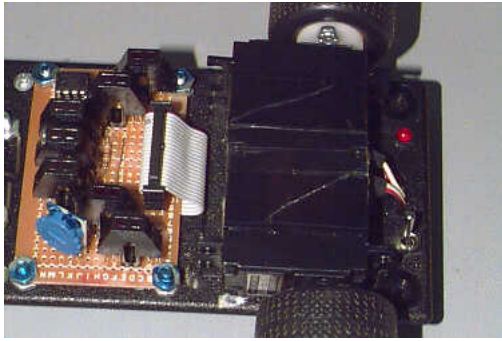
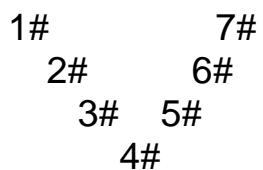


Figure 11: TecBot [14] and JavaBot [15]

WOW! Notice that both of these robots have 7 sensors, but more important, notice the Vee shape of the array. On his site, James Vroman, the builder of these 2 bots only says the “inverted V” allows for tighter turns. But these photos bring up several questions.

Let me digress for a minute. In the back of one of my electronics labs, the students have made a test race track using electrical tape on the white tile floor. One team originally believed that a two sensor robot could navigate a “complex” course with intersections and sharp corners. It just didn’t work. When their robot got to a sharp 120 degree turn, it often missed the turn completely, or partially made the turn, then got confused and oscillated back and forth indefinitely, or once in a while, if it hit the turn just right, it would make it. At a 90 degree intersection, it totally depended on the angle of the robot just before the intersection. If the right sensor got over the perpendicular black line first, it took a right turn. If the left sensor sensed black first, it took a left turn. Only if it hit the intersection exactly square, would it hesitate, but move straight through the intersection. The students gradually came to the realization that more “brainpower” was needed to negotiate complex courses.

At the same time, my brain was in overdrive. As I watched their robot, I pictured an array like this:



In an “easy” course with wide turns, a straight line array would be excellent at sensing and responding to the turns. But a straight line array would be very poor at sensing a greater than 90 degree turn. The advantage that I see in the “V” shown above is that it would be very good at getting an advanced look at intersections and 90 degree turns,

because sensors #1 and #7, out ahead of the normal straight line position, would give an advanced warning.

As always, I'm tempted to go with the "V" array because it seems to have more potential for finding the more complicated turns of an advanced track. And again, I go back to my students. It seems that a straight line array is best for beginners and offers the least complicated patterns for turn detection. So I decide to go with a straight line array for Robot #1. It now looks like I have the following possible configurations:

CONFIGURATION	PROPULSION	SENSOR SHAPE
#1	2 Wheels, independent drives	In Line
#2	2 Wheels, independent drives	"V"
#3	Tricycle, front turning wheel, Rear drive	In Line
#4	Tricycle, front turning wheel, Rear drive	"V"

Table 1: Possible Robot Configurations

It looks like I can get the most learning and most variety by picking robot #1 as a first project and #4 as a second project. I also decide here and realize that I should construct all assemblies with modular design. For example, build the straight line array, but in such a way that it can easily be removed and replaced by a "V" array or an array of any configuration. In programming, I teach the value of modules, and I certainly can benefit from those principles in robot design.

CONSTRUCTING THE IN-LINE ARRAY

With the sensors and the straight line configuration selected, it is time to assemble the array.

The schematic for each sensor looks like this:

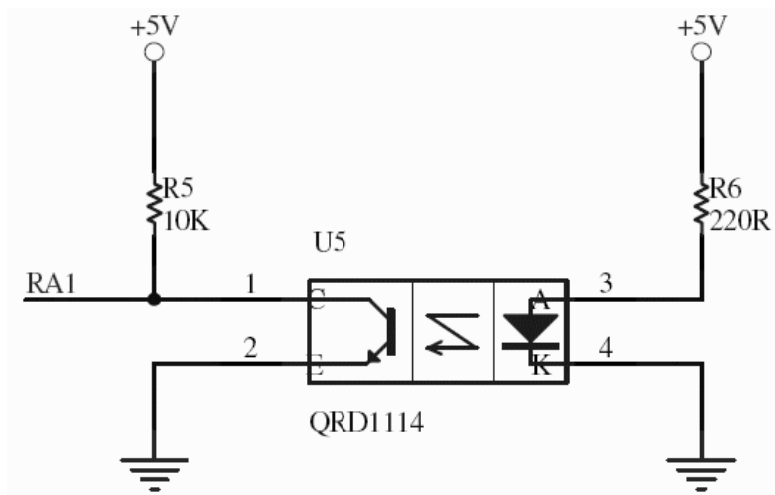


Figure 12: QRD1114 Schematic

On the right the 220 ohm resistor acts as a current limiter to provide the correct current to the LED. Subtracting the 0.7 volts that is normally dropped across an LED, the formula is:

$$I = E / R$$

$$I = (5.0v - 0.7v) / 220 \text{ Ohms}$$

$$I = 4.3v / 220 \text{ Ohms}$$

$$I = 0.01945 \text{ Amps}$$

Or about 20 milliamps, the suggested current for this LED. The data sheet also states that 50 milliamps is the maximum current, and I wanted to keep well away from that.

On the left side, the phototransistor acts as a voltage divider. (It is difficult to see in this figure, but there is a "C" for Collector and an "E" for Emitter on the top and bottom phototransistor leads.) The light reflected back from the floor will hit the base of the transistor, and instead of a base bias voltage being used to control current flow, the amount of light on the base determines current flow. At the extremes:

With no light on the base of the transistor, there is no current flow and it acts like an open switch. With no current flow, a voltmeter put on the output, labeled RA1 in the schematic, would read 5 Volts.

With maximum light on the phototransistor base, the transistor goes into saturation, acting like a closed switch, allowing current to flow from ground, through the phototransistor and resistor. The output will now read 0.0 Volts, since the "shorted switch" makes the output seem to be at the same potential as ground.

And at different light levels, we would expect the output voltage at RA1 to vary between 0 and 5 volts, depending on the light received by the base of the phototransistor.

TEST FIRST!!

One thing I've learned about electronics and programming is that you never want to put anything totally together without finding ways to test and make sure everything is going well throughout the process.

So, I decided to set up one sensor and verify that everything works as expected before making the entire sensor array.

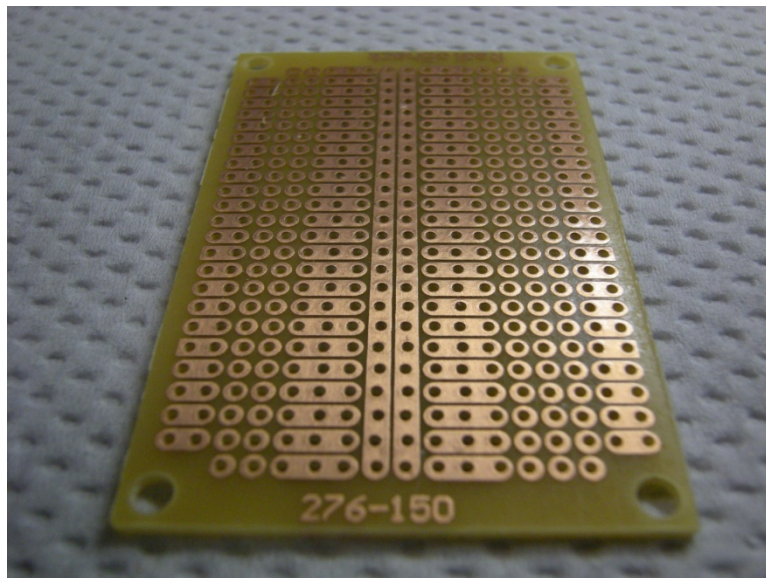


Figure 13: Perf Board

Figure 13 shows a typical perforated circuit board available at electronic suppliers. My first task is to wire up one sensor on this board to test the circuit parameters.

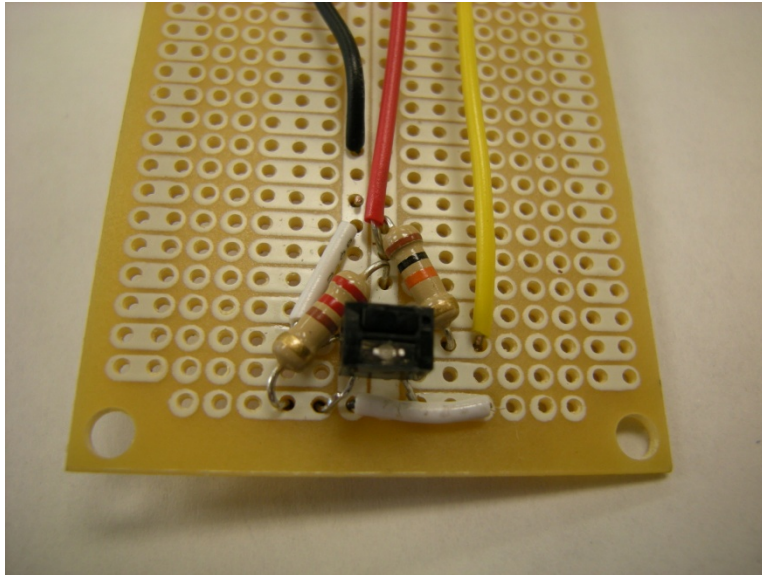


Figure 14: One Sensor Wired

Figure 14 shows one QRD1114 sensor wired as per the schematic above. I then powered this up and sent the output from RA1 to a voltmeter. Next I took a piece of white paper and filled in a $\frac{3}{4}$ inch black line to act as my reflector.

I then held the sensor pointing down and slowly passed it back and forth across the black line to see the voltage change.

With the sensor held more than an inch above the paper, I recorded a fairly constant voltage around 4.2 volts. This confirmed that the “switch was open” or that the phototransistor was not getting any light (except perhaps some ambient light in the room), confirming the high voltage reading for “no reflection”.

With the sensor held about $\frac{1}{2}$ inch above the paper, the change was dramatic. Over the white paper I got consistent readings of about 0.12 volts, confirming that the phototransistor was at saturation and conducting at a maximum. As the sensor was moved slowly over the black line, the voltage jumped nicely to about 4.12 volts, confirming that the black line was not reflecting much light back to the phototransistor, exactly as expected.

I experimented with several heights and found that the sensors worked best (shifting from about 0.12v to 4.1v and back because of the reflectivity of the paper and the line) between $\frac{1}{4}$ inch and $\frac{1}{2}$ inch from the ground/paper. Since there could also be variations depending on the light conditions in other rooms, I decided to make sure I can mount the array in such a way that I could use spacers or otherwise adjust the height of the array from $\frac{1}{4}$ in to $\frac{1}{2}$ inch and maybe a little more.

HOW MANY SENSORS?

Several thoughts led me to the decision to employ eight sensors.

1. My first and most important goal was sensor spacing. Picture sensors spaced at exactly $\frac{3}{4}$ inches or a little more. As the array moves left or right, the most sensors that can be activated (1 = black line is below this sensor) is one. The digital output of eight sensors could only be one of these:

```
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
00000000
```

Which is fine, but I wanted a way to get more precise and accurate measurements. If I space the sensors close enough that when the line is exactly between two sensors, they both react, I can get twice the accuracy, like this:

```
00000001
00000011
00000010
00000110
00000100
00001100
00001000
00011000
00010000
00110000
00100000
01100000
01000000
11000000
10000000
00000000
```

With this arrangement, I get 16 different readings in about the same sensor width where I only got 9 readings in the first example above.

To accomplish this, I settled on $\frac{1}{2}$ inch spacing for the sensors. On a perf board with 0.100 spacing between each hole in the board, I just placed sensor leads 5 holes apart, which you will see below.

2. I wanted the maximum number of sensors I could get. In my Google travels, I'd often seen 6 or 7 sensors used, so that would be my minimum.
3. Lastly, the ports on most microcontrollers have a practical limit of eight, since many ports are numbered something like D0 through D7. Although I've fooled with 10 bit DACs and other odd combinations of bits from 9 to 15, it was clear to me that more than eight of any inputs would require an increased level of complexity. So until I determine that much more complexity is warranted, I decided on 8 sensors. Since 8 sensors also fit conveniently on the standard small perf boards like I was using, that clinched it.

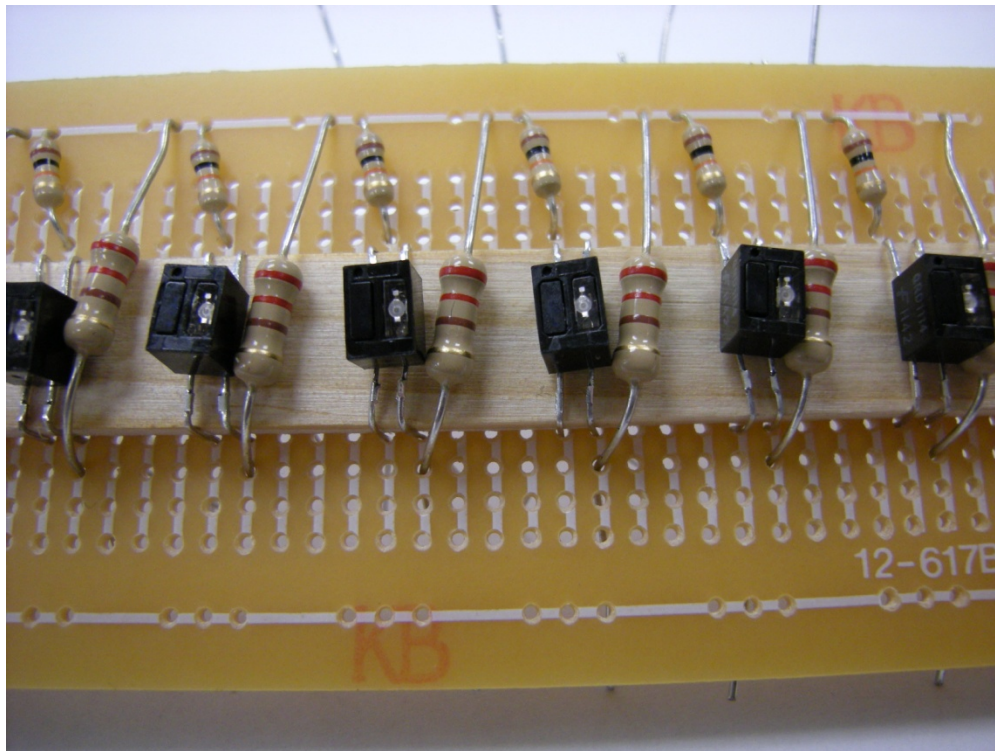


Figure 15: Initial Board Component Placement

Figure 15 shows the sensor board as I first installed the components. Some comments:

- The white line along the top is the 5 volt bus line. Both the 10K Ohm (Brown, black, orange, gold) resistors and the 200 Ohm (red, red, brown, gold) resistors go here.
- The bottom white line, not yet used, will be the ground bus.
- The sensors are all placed in the same direction. LED (small black rectangle) to the left and phototransistor (small round white circle) on the right side.

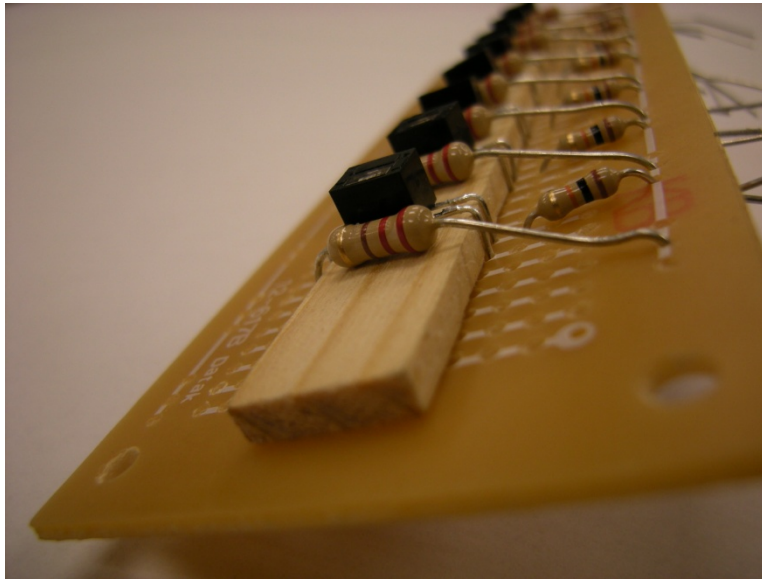


Figure 16: Side View of Sensor array

Figure 16 is a side view of the array showing the piece of wood underneath the QED1114 sensors. The wood is a piece of birch that I picked up at a hobby shop for \$1.50. I would have preferred a popsicle stick (cost = \$0.00).

The wood wound up here due to several considerations.

First – I tried to push the sensors down flat on the perf board. I could do this, but the leads would have entered the perf board with zero length of the leads available. I need at least a little of the lead so that I could apply a heat sink. I've watched my students fry many components because they failed to heat sink properly while soldering. I didn't want that to happen.

Next – I tried to space the sensors about 1/8 to 1/4 inch above the perf board so I could attach a heat sink, but I could not find a way to ensure that all the sensors maintained the same spacing and orientation.

Then the AAAAHA!! Put a popsicle stick under the sensors. That way I could get to the leads with my heat sink and at the same time make sure the sensors were all aligned properly. And the sensors can be seated on the wood to assist in accurate alignment.

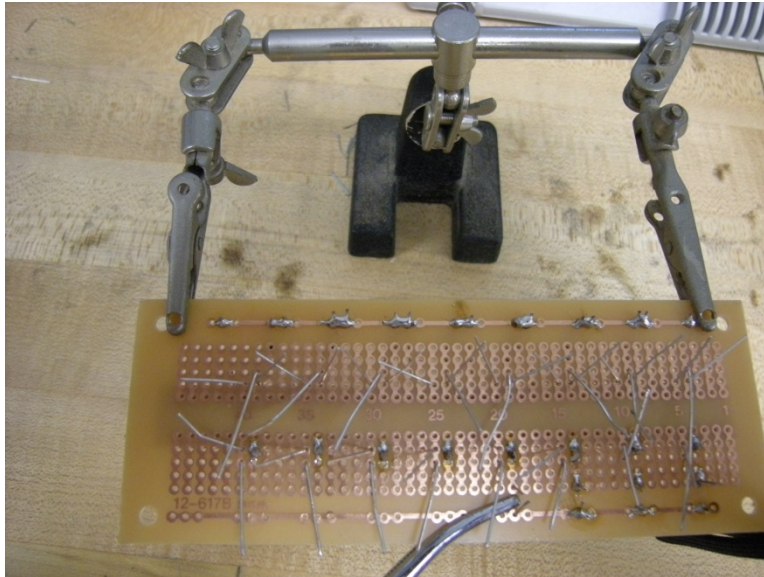


Figure 17: The Third Hand

Since I don't have three or four hands, I found two tools invaluable. Figure 17, top, shows the tool called a "third hand". It clamps the board in just the right position to allow me the angle and approach to complete the soldering. Sticking into the bottom of this figure is the end of a pair of surgeon's forceps that were my main heat sink and often used as a clamp to hold components or wires in place for soldering.

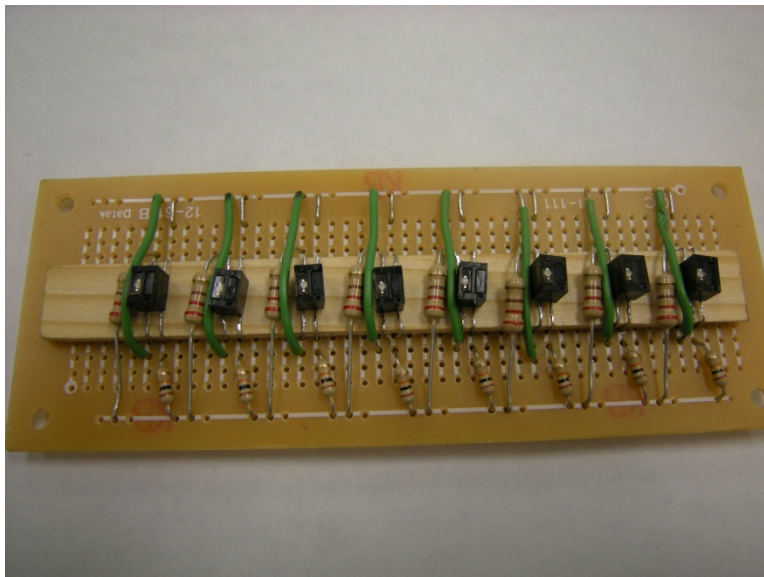


Figure 18: Top Side

Figure 18 shows the top side of the sensor with all components in place and the soldering almost complete. The only thing left is to route the signal leads from each sensor, hook up power and ground and then test the array.

SENSORS COMPLETE

Figure 19 shows the completed Sensor Assembly.

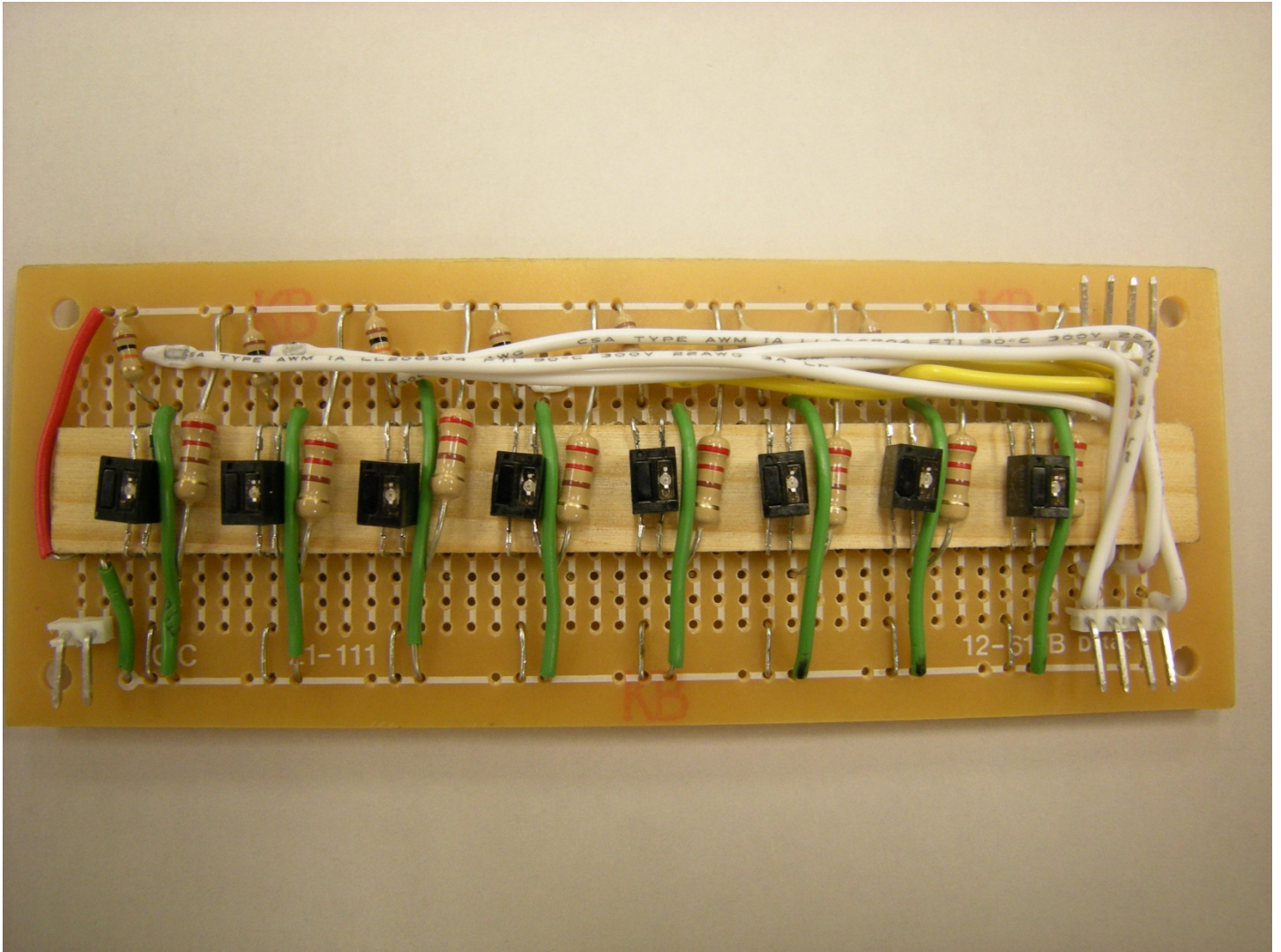


Figure 19: Completer Sensor Assembly

Here is what the complete and tested assembly looks like:

- Top-left Red and bottom-left green leads: Power (5v) and ground, going to a 2-prong 90 degree connector
- White leads: Signal leads for the four left hand sensors, going to a 4-prong 90-degree connector

- Yellow leads (hidden under the white leads): Signal leads for the four right hand sensors, going to a 4-prong 90-degree connector.

Actually the 90-degree connectors I mention above were originally straight connectors. After I soldered them in place I realized that with the board flipped over, the completed connectors would stick out so far that the connector wires would hit the floor or keep me from getting the sensors as close as I desired. Not wanting to unsolder the connectors, I gently bent each of the connector pins over 90 degrees so the connectors when hooked up would stick out to the sides instead of straight down.

I used a DVM to measure the resistance between all of the connector pins. This was to make sure that I did not have any shorts in the system. The readings were normal.

Next I powered up the board, clamped the board into my third hand tool and put a DVM lead on each pin in turn. I then passed a piece of white paper with several black $\frac{3}{4}$ inch stripes in front of the sensors at a distance of about $\frac{1}{4}$ to $\frac{1}{2}$ inch. Each sensor cycled between low and high voltage in reaction to the black stripe as expected. So the sensor array is ready to go.

THE MOTORS

The motors were and probably will be a big challenge for me. My Internet research had shown that David Cook [11] had won contests and achieved fame by breaking the one meter per second barrier in robot line following speed. So the motor part seemed simple. Find a motor that will do better than one meter per second and install it.

The more I learned, the more confused I got. I downloaded about 15 articles discussing motors, torque, stall torque and all the rest. Good theory, but not one piece of advice I could convert into "What motor do I purchase?" I did discover that I probably wanted a gearbox motor running on 6 volts or less. (12 volts or 24 volts = more weight = more mass = harder to make tight turns = slower laps.) I even found an engineering site that had the email addresses for over twenty companies specializing in motors or small motors. I sent a form email letter to all of them explaining all I knew about the robot size, weight, speed, etc. I got back mostly; "We need more information."; or "Stop by our office in Left Snowshoe, Montana and we can discuss this."; or nothing.

I finally had to settle for picking something to see how it works and deciding to learn more and do better for robot #2. I selected the Tamiya Double Gearbox because it had variable speeds. Depending on how you install the gears, you get one of four different gear ratios.



Figure 19: Tamiya Gearbox Motors

I figured that a multi-speed motor would give me more opportunity to experiment and play with gear ratios, speed and torque. Plus Pololu.com had a deal where you could get the robot base, a caster, tires and the motors together for a reduced price, so I jumped on it.

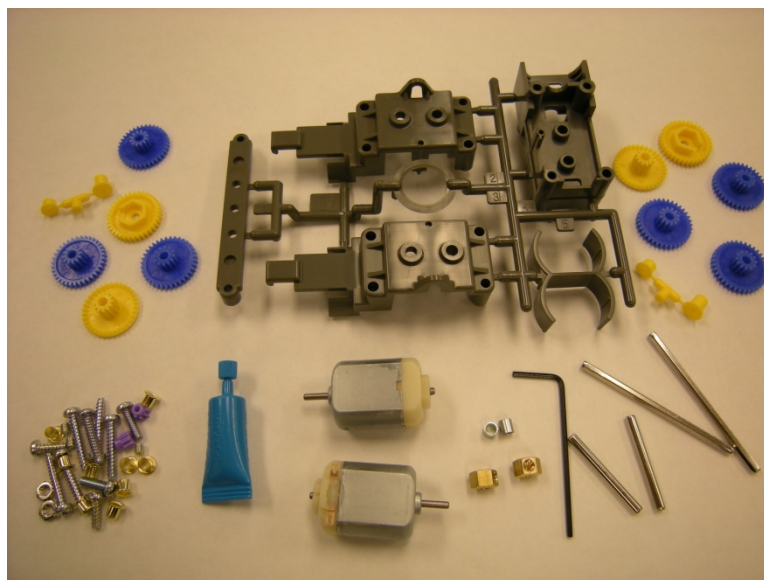


Figure 20: Tamiya Gearbox Motor Parts

As soon as I saw all of the non-English writing on the directions, I almost freaked out. I've read several sets of so-called "English" directions from Far East companies and they were a nightmare. Fortunately, the directions turned out to be clear, concise and

easy to follow. Except for the fact that you need three or maybe four hands to hold and balance all the parts and gears just before they all come together, I was pleased with the result.

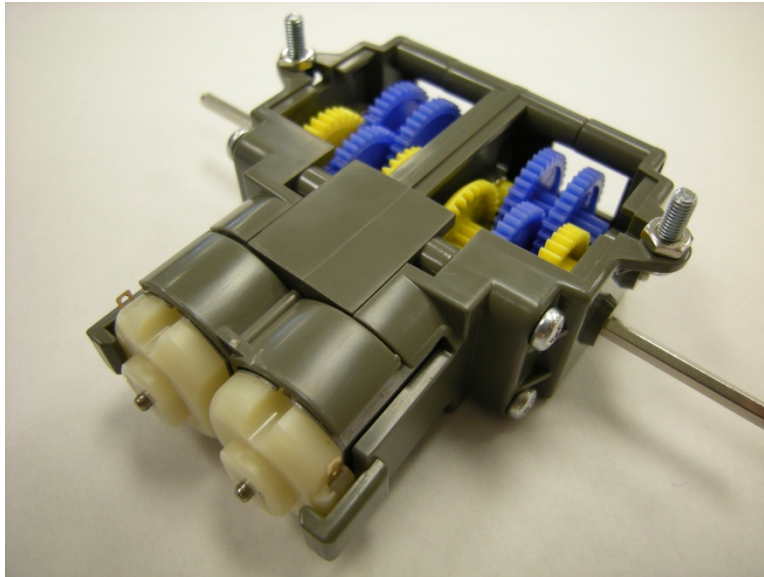


Figure 21: Assembled Motors

I had to be careful to put all the parts and directions in one baggy and clearly mark it so I can change the gear ratios later if I need to.

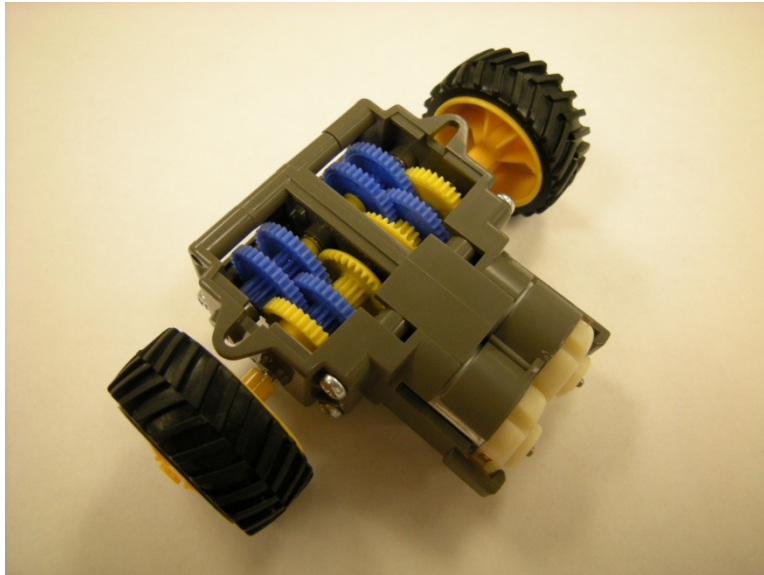


Figure 22: Motors with Wheels Attached

And as Figure 22 shows, just pop on the wheels and the motor assembly is complete.

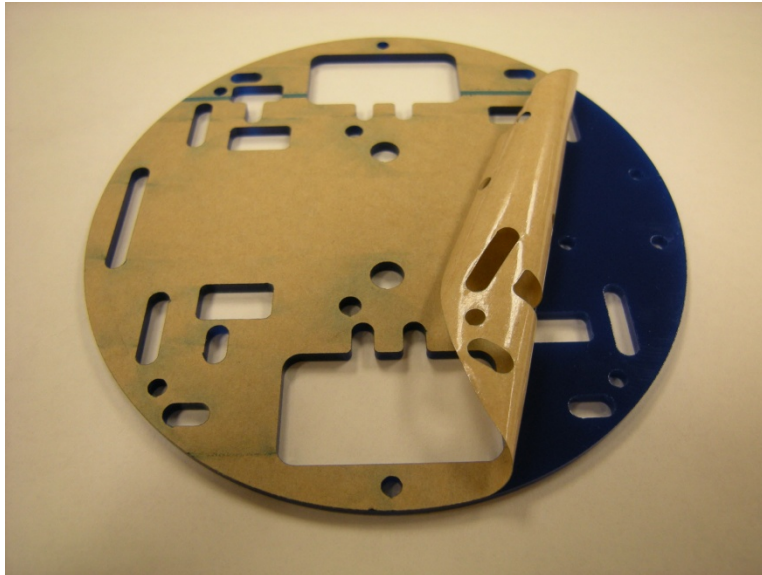


Figure 23: The Robot Base

Figure 23 shows the circular, plastic base I selected for the robot. Just peel off the protective paper and it is ready.

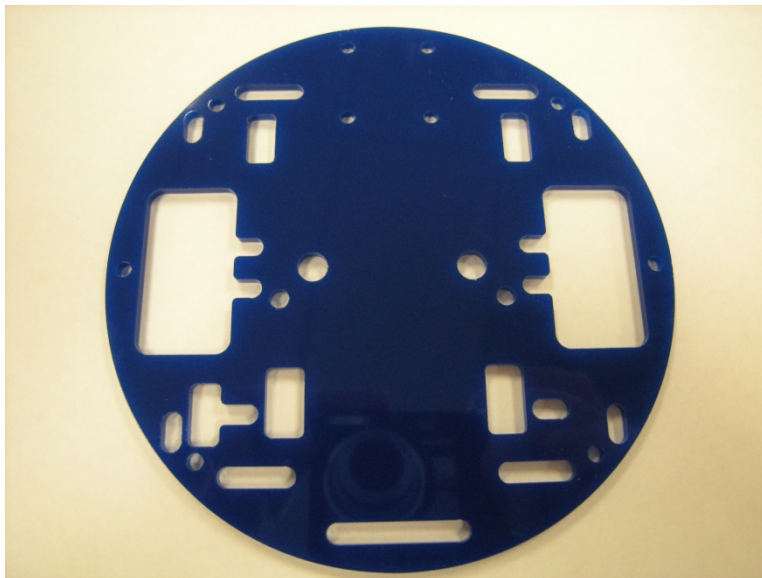


Figure 24: The Robot Base Revealed

Notice in figure 24 that the robot base already has a number of holes cut to make your assembly job easier. The large rectangular holes, left and right are for the wheels and the slots to the inside for the wheel mounting hardware. The four holes in a square at the top are for the caster which will get assembled soon.

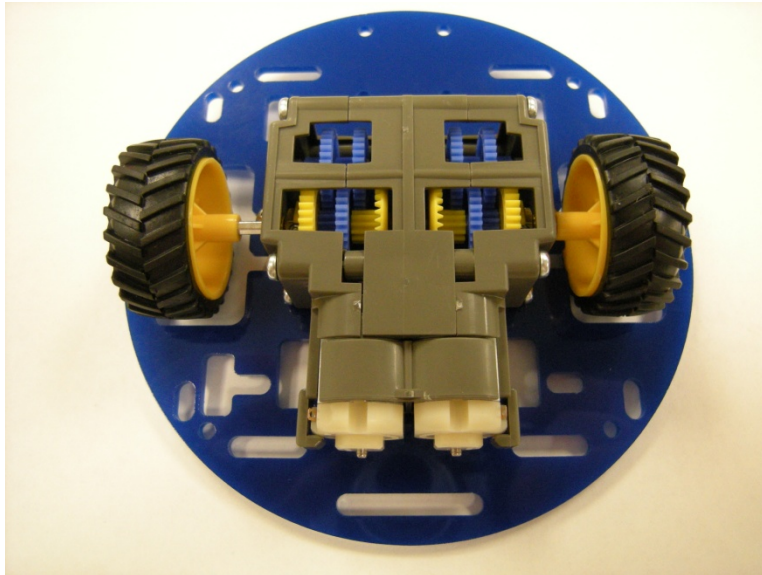


Figure 25: Motors and Base

It is a simple matter to join the base and motors with two small nuts and bolts.

THE CASTER

Figure 26 shows the caster I purchased. I have used swivel wheels before and I was never impressed. They always seem to hang up or get stuck at the worst possible times. With a line following robot that had to react fast and never get hung up by a swivel wheel, I decided to go with the caster.



Figure 26: The Ball Caster



Figure 27: Caster Parts

The caster, as expected, was a simple task. The only delicate part was placing the three bearing rods above the caster wheel as shown in the next two figures.



Figure 28: Bearing Rods Before Placement



Figure 29: Bearing Rods After Placement



Figure 30: Caster Bottom Assembled

Once the bearing rods are in place, a few snaps, three screws and it is all together. The pieces even allow different alignments so that your caster height can be set high or low.

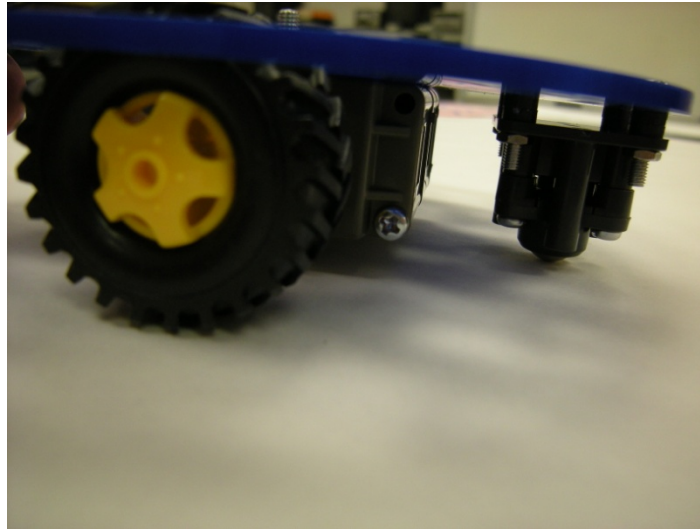


Figure 31: Assembled Caster, Side View

The caster is a simple mount to the robot base.

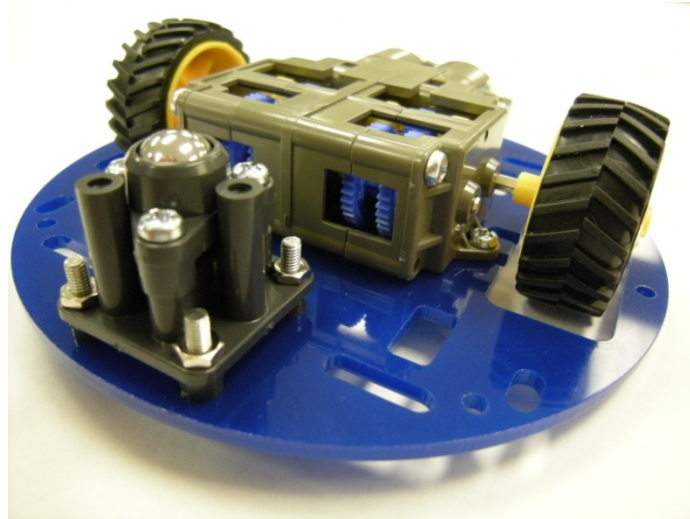


Figure 32: Robot Base, Bottom View

And Figure 32 shows the completely assembled robot base, upside down.

THE NEXT INSTALLMENT

So, having made some basic decisions about how to proceed, a future article will continue the robot design and construction process. Soon we will know how fast it can go.

I also intend to track my students as they complete the Embedded Systems class and report how they are doing with line following, line mazes or other PIC robotic projects.

Your comments and suggestions are welcome at RVannoy@ITT-Tech.edu.

I am also just beginning to set up a web site devoted to PICs and Robotics. Although the site is not ready as of mid-February, 2008, I hope to have some interesting and informative content soon at <http://www.PICRobots.com>. Feel free to stop by and visit.

Photo Sources:

[1] Chris and Dawn Schur's Robotics and Artificial Life Forms,

<http://www.schursastrophotography.com>

[2] Wallace, David N., Line Following Robot <http://www.lifekludger.net/category/weekly-links/page/2/>

[3] Denmark's Techniske Universitet <http://www.sweeper.org>

[4], [9] Mike's Line Following Robot, Central Illinois Robotics Club,

<http://www.circ.mtco.com>

[5] <http://www.leang.com>

[6] <http://www.cs.umn.edu>

[7], [8] Jackson, Ben, <http://www.ben.com>

[10] <http://www.ece.unm.edu>

[11] David Cook's Jet <http://www.robotroom.com/jet.html>

[12] <http://blog.makezine.com>

[13] The CBA Line Following Module <http://www.budgetbot.com>

[14] <http://www.james.vroman.com/tecbot1a.htm> james@vroman.com

[15] <http://www.james.vroman.com/javbot1a.htm>