Line Following Robot

Frank Bruno James Ihrig

University of Central Florida University of Central Florida  
4000 Central Florida Blvd 4000 Central Florida Blvd

Orlando, Florida, 32816 Orlando, Florida, 32816

fbruno13@knights.ucf.edu JEIhrig@gmail.com

Abstract

This project implements a line following robot. This robot uses a PID controller, a feedback control loop, implemented in an Atmel 8 bit microcontroller to read in the location of a line from an array of IR led/photodiode pairs. Using this reading from the sensors, an error term is calculated. This error term is then used to create a turn rate, which tries to lead the center of the robot back to the line. The values used to calculate this turn rate are P, I, and D. These values are constants that must be tuned to a good value using trial and error. This project outlines an automated learning method that can be used to find good values of P, I, and D in a semi-supervised fashion. This method uses default good values found by short manual tuning and uses simulated annealing to try different PID values for a lap around a defined track. Using a defined performance measure, the different values of PID are compared in real time.

# Introduction

A line following robot is a simple robot design that uses sensors to make small adjustments to the motors in order to follow a line on an opposite contrast surface. This line following robot happens to read a black line on a white surface, as seen in Figure 1. This line following robot aims to achieve a high speed, low error, and low cost solution to following any line laid out by a user.



Figure – The line following robot and the track it will be running on.

This line following robot makes use of a PID controller to follow the line using input from an array of IR led and photodiode pairs. The design of the hardware was simplified to using many off the shelf components to allow for fast prototyping and a focus on the software related to a PID controller. In addition, this project aims to introduce learning to the tedious process of manually tuning a PID controller.

The motivations for having this automatically tunable PID controller are many. It would allow any speed motor to be trained without having to rely on manual methods that could take a very long time to learn in a supervised environment. The automatic method that is purposed here is a semi-supervised method with the user having to lay out the track and specify a start position using a surface consisting of a black line that is long enough to hit all of the sensors. The robot is designed to stop if it runs off the track at which point the user must manually place the robot back on the track.

# Related Work

There are various methods of line following robots out there. Many of them are the same, following the time old tradition of using IR led /photodiode pairs to act as the eyes of the robot. A very recent paper submitted to the ICCAE done by Pakdaman et al. shows the design and implementation of a simple line following robot using just this method. The benefits of using this design are it is simple, cost effective, and reliable. Some have improved upon the reliability of the IR led/photodiode pair by using an IR emitter and an IR NPN phototransistor, which are very cheap and more reliable than the original pairing. Others, like Dupuis and Parizeau, have successfully used cameras. Each of these sensors has their own advantages and disadvantages.

Most work done on line following robots use a low cost, low power microcontroller to do the job of reading in values from sensors and controlling the motors. Pakdaman et al. uses this method. Another method used by some is a transistor method that doesn’t use microcontrollers at all. This method relies on a hardware only approach to follow the line. A disadvantage of this is has no method of learning if the implementers would like a better line follower.

Various control techniques have been used to control the robot to do line following. The simplest of them is the PID controller. This method uses a feedback control loop to adjust the turn of the wheels to follow the line. For this method, manual tuning must occur to get a good line follower. Other methods include Collins and Wyeth’s method of using a Cerebellar Network to learn the correct turn from the sensors and Dupuis and Parizeau’s method of genetic programming to evolve a control method.

# Method

This section focuses on the implementation of the line following robot. It consists of the hardware design description, PID controller, and the PID learning algorithm.

Hardware

The focus of this section is on the design decisions made while constructing the robot such as what parts were used in creation of the robot including sensors, microcontroller (MCU), Wireless Command and Reporting, Power, and Motors. It will also cover some of the hardware limitations and the cost of the robot.

Chassis

As discussed in the related work, many line following robots are small RC car like robots. They can have 2 or more wheels. The smaller and lighter the robot the more speed the robot will have. The body of the line following robot will consist of a Pololu 5" Robot Chassis RRC04A Solid Blue base with a Pololu 3pi Expansion Kit without Cutouts – Black mounted to it 1.5” above. The expansion kit is essentially a prototyping board with slightly smaller dimensions of the base. This chassis allows for the most flexibility to design the rest of the robot and is very light weight. Additionally, the base comes with cutouts for the wheels and many predrilled holes. This design does limit the amount of wheels to 2, but given the light weight design 2 wheels should be sufficient. Since the robot is using only 2 wheels, a Pololu Ball Caster with 3/8" Metal Ball will be used with a counter weight to keep the robot balanced. The ball will just roll as it makes contact with the surface.

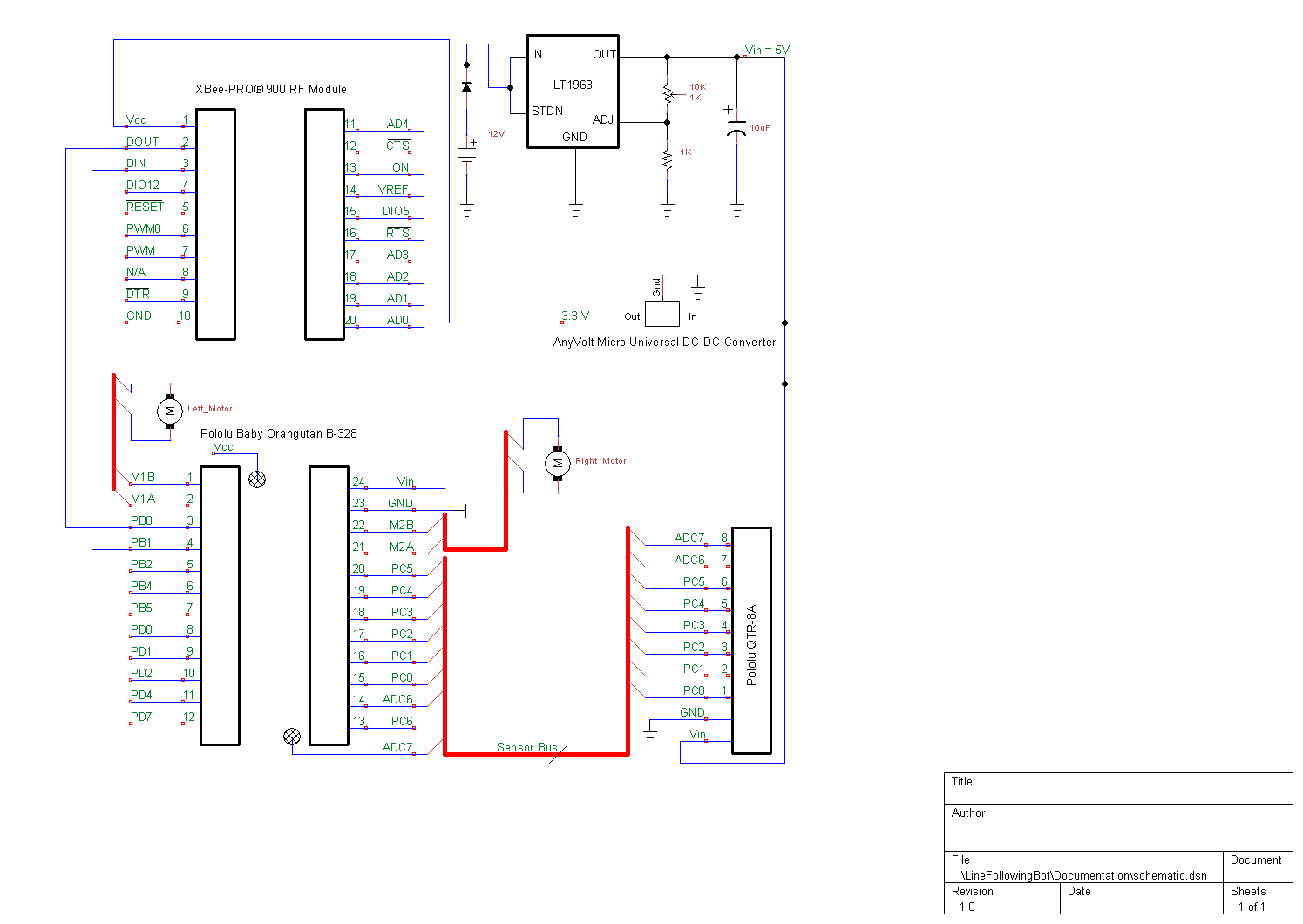


Figure - Pololu QTR-8 Sensor Array

Sensors

As discussed in the related work, many line following robots simply use some form of an IR led and a Photodiode used to measure the reflectance from the surface directly below the robot. This combination will output a voltage based on the level of reflectance from the surface. This voltage will be close to zero on a white surface and close to the voltage of the input source on a black surface. A very important choice that needs to be made is how many IR led and Photodiode pairs (sensors) will be used. This is mainly dependent on the size of the robot and the width of the tape. Ideally, the amount of sensors is as many as possible to spread across the width of the robot without the sensors interfering with each other. Since the track is made using 0.75” inch electrical tape, see Figure 1, the ideal width is smaller than the tape. This way more than one sensor would be able to see the tape at a time. Also, the sensor array shouldn’t be much longer than the robot, which is 5 inches. Given these specifications, the ideal amount of sensors is 8 or at the worst 6.

With these specifications in mind, there are two easy choices for the sensor array the Pololu QTR-8A Reflectance Sensor Array, shown in Figure 2, and Pololu QTR-8RC Reflectance Sensor Array. With a good price point of only 15 dollars each, both are very good choices. The QTR-8 comes with 8 IR/Photodiode pairs and can be run at either 5 or 3.3 volts, very common voltage ratings. The spacing between each sensor is 0.375 inches. This falls in line with the required specifications. The only difference between them is the RC can be hooked up to any digital I/O and the A must be hooked up to an analog input only. Since either of them are good choices the choice of the MCU will determine, which sensor array will be chosen based on the number of analog inputs.

Motors

The motors are a very important since they will be the driving force propelling the robot forward. An important goal of this project is to create a line following robot that is very fast. Therefore, a variety of motors were researched from Pololu. All of these motors have a voltage rating of 6 volts, but will work with any voltages between 3 and 9 volts. So, they are compatible with the already chosen sensor array. Additionally, these motors are small 0.94" x 0.39" x 0.47" and 0.35 oz. The first of which is a 50:1 Micro Metal Gearmotor HP with Extended Motor Shaft. This motor is very fast with 625 RPM, they use 100 mA when free running and 1.6 A at stall. This motor has 15 oz-in of torque. The second is a 50:1 Micro Metal Gearmotor with Extended Motor Shaft. This motor is much slower than the previous with only 250 RPM, but they use just 40 mA when free running and 360 mA at stall. This motor has 6 oz-in of torque. The last motor reviewed is a 100:1 Micro Metal Gearmotor with Extended Motor Shaft. This motor is the slowest of the motors reviewed with 120 RPM, they use 40 mA when free running and 360 mA at stall. This motor has 10 oz-in of torque. Although, they are the slowest they provide more torque than the 50:1 standard motors. The ideal choice is the 50:1 HP motors because it meets our specifications better than the others, but it requires a power source that can provide up to 3.2 A if the wheels stall, and a MCU that can output at least 100mA and up to 1.6 A per motor. Since Pololu offers brackets and wheels that will fit any of the motors reviewed, the wheels for the line following robot were chosen to be a pair Pololu Wheels which are 42x19mm and the pair of brackets chosen were the Pololu Micro Metal Gearmotor Bracket Extended.

Robot Controller

With the sensors and motors chosen, the next choice is what will receive the input from the sensors and use this data to drive our motors to achieve the goal of line following. The Pololu Baby Orangutan B-328P was chosen as our robot controller, shown in Figure 6. This decision was based on the fact that this robot controller uses an 8-bit Atmega328p microcontroller in a 24-pin form factor complete with headers for an easy to manage prototyping environment, which is ideal given that a prototyping board is being used. The Atmeg328p comes with the ability to drive two independent motors with a continuous current supply of 1 Amp per motor with a peak current at 3 Amps per motor. This falls in line with any motor that was discussed. Therefore, the 50:1 motors will be used since they provide the most stable prototyping environment with medium velocity. In addition, this MCU comes with 8 analog inputs, which is exactly how many we need for the 8 sensor inputs. Therefore, the QTR8-A sensor array will be used. Also, the recommended input voltage is between 5 – 13.5 volts.

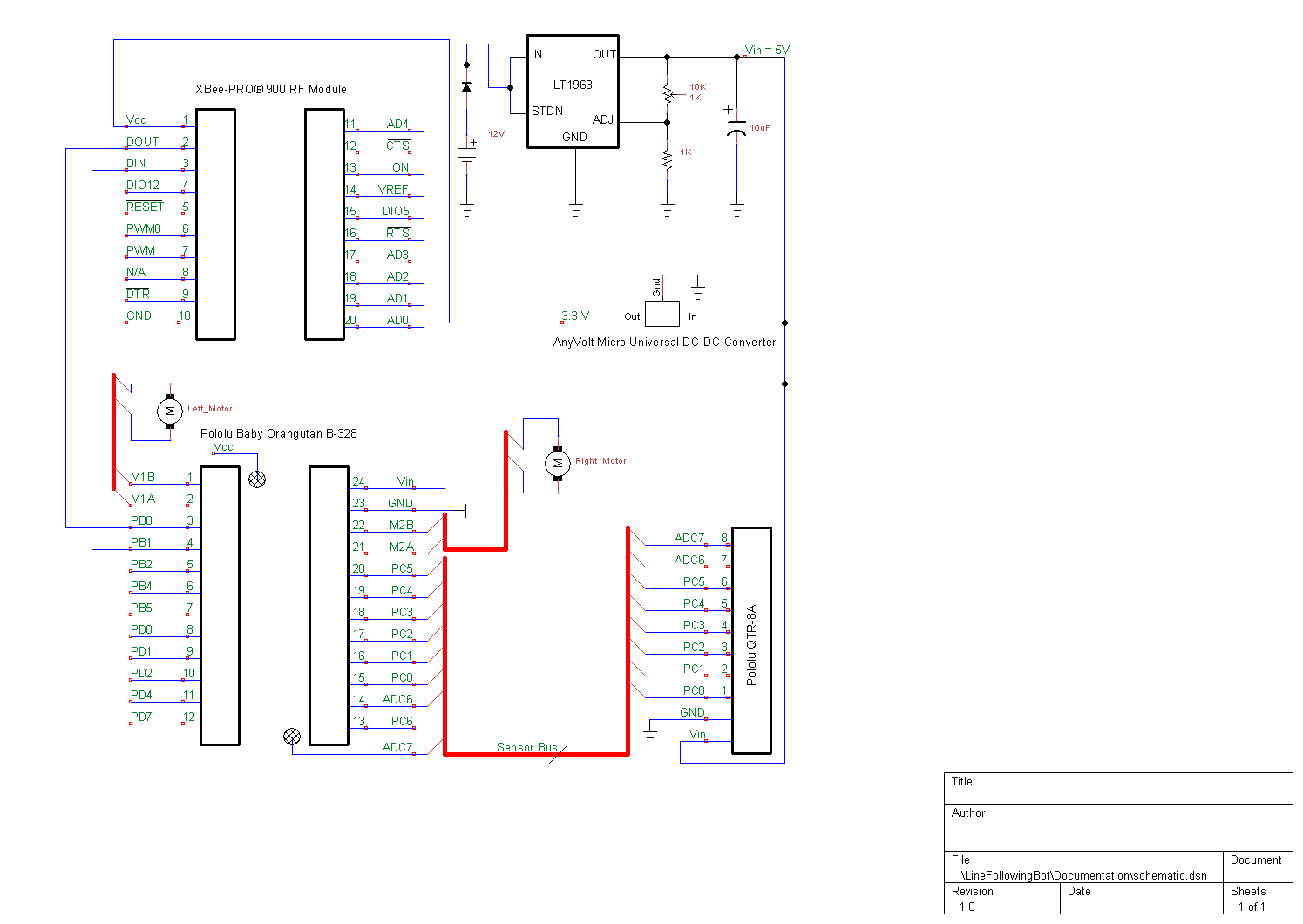
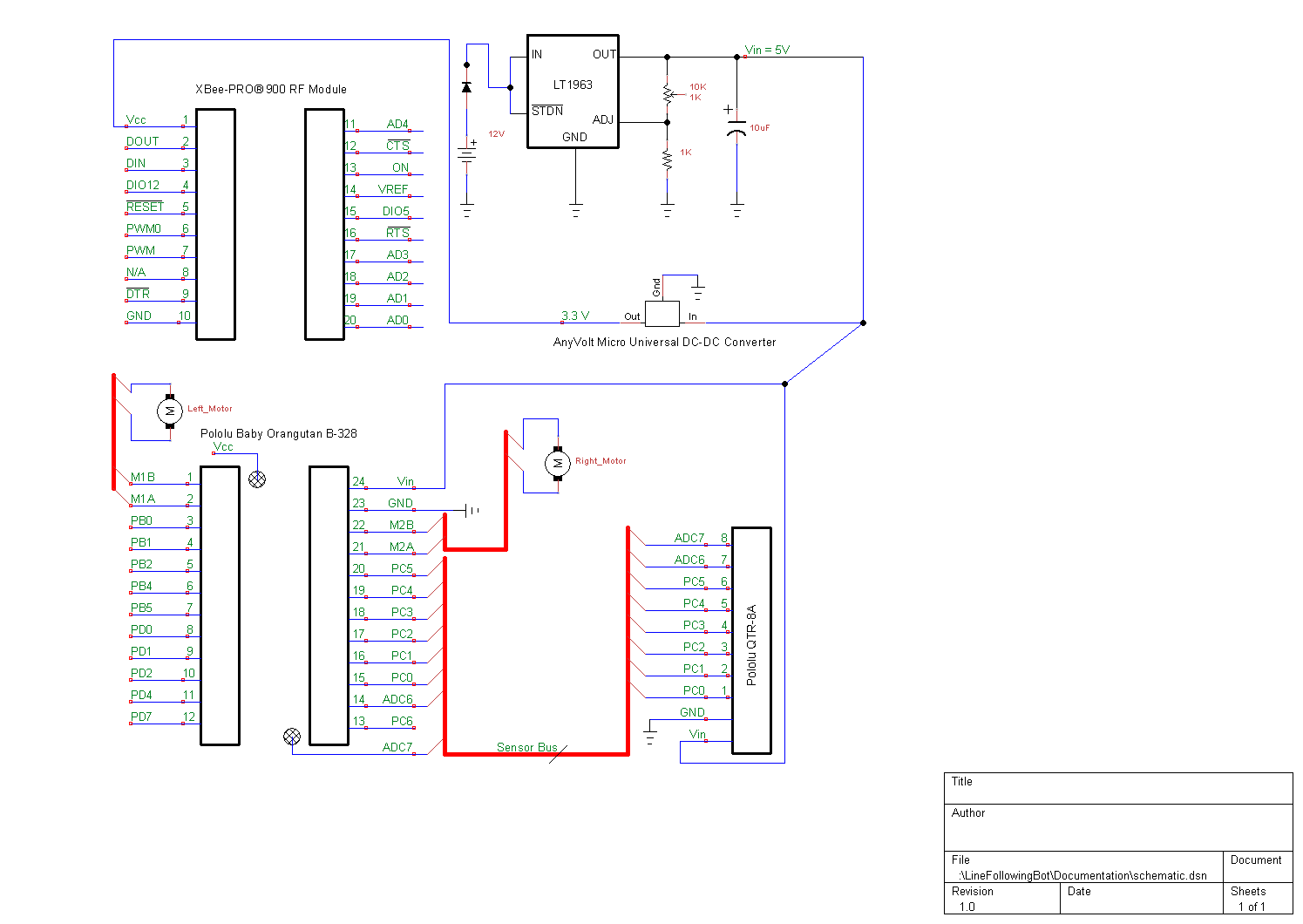


Figure - Pololu Baby Orangutan B-328P for full schematic see Appendix A. Figure 6

Power

In order to power these devices, a power supply must be chosen. Additionally, the voltage must be regulated using a voltage regulator. This insures no spikes from a battery will cripple any of the components. Since all of the components have support for an input of 5 volts, 5 volts is used as the voltage for the robot. The LT1963ET voltage regulator from linear technologies is used to do the job of regulating the voltage, the schematic is shown in Figure 4. It can receive input voltages of ± 20 V and supply 1.5 A to the robot at any voltage from 1.21 V to 20 V. Our ideal choice for a battery is a 9 V, but due to the large supply of current required from the motors a single 9 V cannot be used. So, instead the robot uses a 12 V Thunder Power Rechargeable Li-Polymer battery with a capacity of 2070 mA/h. This allows the robot to run for a few hours and a rechargeable battery saves buying new batteries.

Figure – Voltage Regulator



Wireless Command and Reporting

The wireless command and reporting feature is an optional feature that helped the debug process and created a more accurate evaluation this robot. A pair of XBee-PRO 900 extended range modules were used to communicate between a laptop and the robot. This allowed for on the fly debugging and the reporting of run data including the total error for that lap around the track. This could also be used to send commands to the robot wirelessly. Many of the uses of this feature will be discussed further in the evaluation section of this paper. Unfortunately, these modules only run at 3.3 V and in order to use this module an AnyVolt Micro Universal DC-DC Converter is used to bring the output voltage down to 3.3 V for this part only. Since this is an optional feature the cost of these parts will be deduced from the total cost to build, but the price will be include for reference.

Cost

The cost of each of the major components is listed in and the optional components are listed in .

|  |  |  |  |
| --- | --- | --- | --- |
| Product | Cost | Quantity | Total Price |
| Pololu Wheel 42x19mm Pair | $6.98 | 1 | $6.98 |
| Pololu Micro Metal Gearmotor Bracket Extended Pair | $4.99 | 1 | $4.99 |
| Baby Orangutan B-328 + USB AVR Programmer Combo | $31.95 | 1 | $31.95 |
| Pololu 5" Robot Chassis RRC04A Solid Blue | $7.95 | 1 | $7.95 |
| Pololu Ball Caster with 3/8" Metal Ball | $2.99 | 1 | $2.99 |
| 3pi Expansion Kit without Cutouts - Black | $19.95 | 1 | $19.95 |
| LT1963ET | $3.00 | 1 | $3.00 |
| QTR-8A Reflectance Sensor Array | $14.95 | 1 | $14.95 |
| 50:1 Micro Metal Gearmotor with Extended Motor Shaft | $16.95 | 2 | $33.90 |
| Grand Total |  |  | $126.66 |

Table – Bill of Materials

|  |  |  |  |
| --- | --- | --- | --- |
| Product | Cost | Quantity | Total Price |
| AnyVolt Micro Universal DC-DC Converter | $19.99 | 1 | $19.99 |
| XBee-PRO 900 extended range module w/ wire antenna | $39.00 | 1 | $39.00 |
| 50:1 Micro Metal Gearmotor HP with Extended Motor Shaft | $16.95 | 2 | $33.90 |
| 100:1 Micro Metal Gearmotor with Extended Motor Shaft | $16.95 | 2 | $33.90 |

Table – Optional and substitute parts

PID controller

The robot uses a well know method known as a PID controller to control the motors. PID stands for Proportional Integral Derivative. A PID controller works based on current error, duration of error and change in error and tries to counter such error in a balanced way that results in a smooth accurate tracking. When deciding to use a PID controller, the first thing that must be decided is how to calculate the initial error. For the robot, the error is calculated by checking if each of the sensors is activated (above some threshold.). Specifically, it can be described as:

NUM\_LINE\_SENSORS = 8

DESIRED = 0

for ( i = 0; i < NUM\_LINE\_SENSORS; i++)

{

lineSensor[i] = analogRead(i)

if (lineSensor[i] > SENSOR\_THRESHOLD)

{

if (i < NUM\_LINE\_SENSORS/2)

{

actual -= (NUM\_LINE\_SENSORS/2 - i)

}

else

{

actual += (i - NUM\_LINE\_SENSORS/2 - 1)

}

}

}

error = DESIRED – actual

Where linesSensor[i] can range between [0, 1023], depending on activation level, and SENSOR\_THRESHOLD is set to some value in the range [0, 1023], in this robots case it is 400. A simplified way to look at this is to check if each sensor is activated, if it is, then add to error if it's on the right side, subtract if it's on the left side.

Now with a definition for error, construction a PID controller can begin. The proportional component of the PID controller tells the robot to turn proportionally to the error., the integral component tells the robot to use the sum of error over time to determine how much to turn when current correction has not been enough to correct the error over time, and the derivative component tells the robot uses the rate of change of the error to dampen the turn rate to help prevent over-correction. Each of the terms is multiplied by some constant to tune each constant until we get good results. First, a P value only was used to test on the track. It was tweaked it until it seems to track with minimal oscillation. Then, a D term was added to dampen some of the oscillation. The I term was then added to ensure that the robot does not take turns too widely.

Initial testing has found some constants for P, I and D that yields some good results. For a more formal test, a small range of values near those found will show the differences that are seen when other values are used. In the tests so far, the best constants for P, I and D we have found are 50, 0.05, and 10 respectively.

Here is the code used for our PID controller:

P = 50, I = 0.05, D = 10

//The P multiplicand

error = desired - actual;

//The D multiplicand

dError = 2(error - oldError) / (currentTime() - lastFrameTime));

//The I multiplicand

sumError += error \* (currentTime) - lastFrameTime);

//Set old error to current

oldError = error;

lastFrameTime = currentTime();

//Turn rate

return P \* error + I \* sumError + D \* dError;

Learning PID values

Many different ways to learn PID values have been considered ranging from Genetic Algorithms, Neural Networks, to simple gradient descent. After realizing the lack of a simulated environment, large degrees of exploring will be detrimental to the leaning process as it will cause the robot to go off course and there is have no way of automatically resetting. Therefore, the search has been mostly focused on using different forms of gradient descent that don’t explore much, such as the hill climber algorithm and a custom tree method created for this robot, but the hill climber seemed too restrictive and the tree method would either use too much memory, or lose some good values. In the end, simulated annealing was decided to be used to learn the values as it seemed to be the best option for searching from a known decent value and keeping memory usage low.

The implementation of simulated annealing requires some way of evaluating performance of each run. To do this a score will be assigned to each run by adding the lap time to the total error seen during the run. With this measure, runs with greater values are considered worse runs. With simulated annealing, the initial values of P, I, and D are known decent values and it explores by randomly selecting new values of P, I and D within a predefined range. Determined by:

Where is the total number of runs to do before completing training, is the current run the robot is on, and each of the are the range of values that the new value can vary away from the best value. For the simulated annealing algorithm, a range of values are set that the program can choose from for P, I and D in each run. If it chooses values that perform better, it updates the best known values and any new values chosen will be centered near the new value. If it chooses values that perform worse, those values are thrown out. If values are chosen that are so bad that the robot cannot stay on the track, it receives a penalty of 100 each iteration plus any extra time that passes while robot is placed back on the path and is thus guaranteed not to be a candidate for the best run. Each run, the range of values it can choose from narrows around the current best, eventually converging at some set of values for P, I and D. The more runs that are given, the better the convergence will be. Due to memory constraints, the robot can only track 20 values to store it its history. The algorithm does not require us to store these values, but it has to be done to show results from the run, since outputting serial data while running causes the robot to drop samples and become extremely inaccurate. A piece of black tape has been placed on the track tangent to the course. When the robot crosses this mark all sensors are activated simultaneously. This allows the simple state machine to know when to save off the lap time and start recording a new lap. Each new lap starts a new iteration of the annealing algorithm, which is designed to converge once all laps have been completed.

# Evaluation

This project is evaluated with respect to different values of P, I, and D and the performance score received for an average of 5 laps around the track. The performance score is based on speed contributing 50% and the total accuracy being 50%. Both the accuracy and the lap time are defined as a lower is better value. These values are recorded by the microcontroller and reported back to the laptop using the wireless serial communication module.

The first set of tests is done on values set from observable good runs done during the initial testing of this robot. These values and the performance with respect to error rate and speed are shown in Table 3. Additionally, the values are compared with the values from automated learning.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Run | P | I | D | Error | Time | Score |
| 1 | 40 | 0 | 0 | 4447 | 8.274 | 6361 |
| 2 | 40 | 0 | 10 | 4600.2 | 8.259 | 6430 |
| 3 | 40 | 0 | 15 | 4109 | 8.309 | 6209 |
| 4 | 40 | 0.05 | 0 | 3917 | 8.173 | 6045 |
| 5 | 40 | 0.05 | 10 | 3814.8 | 8.202 | 6008 |
| 6 | 40 | 0.05 | 15 | 4021.8 | 8.210 | 6116 |
| 7 | 40 | 0.1 | 0 | 3372.6 | 7.964 | 5668 |
| 8 | 40 | 0.1 | 10 | 3552 | 8.3 | 5926 |
| 9 | 40 | 0.1 | 15 | 3468.8 | 8.380 | 5924 |
| 10 | 50 | 0 | 0 | 3579.4 | 7.964 | 5772 |
| 11 | 50 | 0 | 10 | 3712 | 8.308 | 6010 |
| 12 | 50 | 0 | 15 | 3711 | 8.416 | 6064 |
| 13 | 50 | 0.05 | 0 | 3631.6 | 8.292 | 5962 |
| 14 | 50 | 0.05 | 10 | 3414.4 | 8.332 | 5873 |
| 15 | 50 | 0.05 | 15 | 3503 | 8.366 | 5935 |
| 16 | 50 | 0.1 | 0 | 3294 | 8.311 | 5803 |
| 17 | 50 | 0.1 | 10 | 3270.2 | 8.487 | 5879 |
| 18 | 50 | 0.1 | 15 | 3819.8 | 8.598 | 6209 |
| 19 | 60 | 0 | 0 | 3498.6 | 8.293 | 5896 |
| 20 | 60 | 0 | 10 | 3812.4 | 8.618 | 6215 |
| 21 | 60 | 0 | 15 | 3963.6 | 8.488 | 6226 |
| 22 | 60 | 0.05 | 0 | 3062 | 8.197 | 5630 |
| 23 | 60 | 0.05 | 10 | 3613.2 | 8.731 | 6172 |
| 24 | 60 | 0.05 | 15 | 3676 | 8.746 | 6211 |
| 25 | 60 | 0.1 | 0 | 3498.6 | 8.293 | 5896 |
| 26 | 60 | 0.1 | 10 | 3252.6 | 8.450 | 5851 |
| 27 | 60 | 0.1 | 15 | 3287.2 | 8.563 | 5925 |

Table – Experiments with Different PID Values

Comparison between Controllers

One of the goals of this project is to produce a low error, fast time around the track. The goal of the previous experiments was to show the differences between the values of P, I, and D. The results are very interesting. Table 4 shows the top ten runs as determined by the scoring previously defined. The first interesting point is 3 of the top 5 runs have only the P and I component. The second interesting point is that a P controller out performs the PID controller. It appears that the slower the speed of the robot the less the D term is needed. Additionally, not a single PD controller ranked in the top 10. It seems that for a robot this small, lightweight, and slow that the P and I term only would suffice for good lap times with lower error. It would be interesting to see the manual tuned values for a robot that had faster motors. This is an area for improvement.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Run | P | I | D | Error | Time | Score |
| 22 | 60 | 0.05 | 0 | 3062 | 8.197 | 5630 |
| 7 | 40 | 0.1 | 0 | 3372.6 | 7.964 | 5668 |
| 10 | 50 | 0 | 0 | 3579.4 | 7.964 | 5772 |
| 16 | 50 | 0.1 | 0 | 3294 | 8.311 | 5803 |
| 26 | 60 | 0.1 | 10 | 3252.6 | 8.45 | 5851 |
| 14 | 50 | 0.05 | 10 | 3414.4 | 8.332 | 5873 |
| 17 | 50 | 0.1 | 10 | 3270.2 | 8.487 | 5879 |
| 19 | 60 | 0 | 0 | 3498.6 | 8.293 | 5896 |
| 25 | 60 | 0.1 | 0 | 3498.6 | 8.293 | 5896 |
| 9 | 40 | 0.1 | 15 | 3468.8 | 8.38 | 5924 |

Table – Top 10 runs

Comparison of PID Values

During the initial runs without learning we have found some values of P, I, and D that work well and others that do not. The learning algorithm has been tested with known good values and known bad ones. During runs with known good values, most changes during the run are vaguely noticeable but a difference can be seen between some runs. Starting it with known bad runs such as a high P value, the robot is very shaky as it tries to over-correct it's error but its over-all error and speed are still quite good so it takes many iterations for it to learn a lower P or higher D to smooth out the shakiness. When started with a P that is too low, it cannot stay on the track most of the time and must be guided by hand. However, due to the penalties in place for leaving the track, it never learns any values that are worse than the current, and will fairly quickly begin to learn values of P that allow it to stay on the track without intervention.

After many short run trails, a long training session was tested. It consisted of 100 laps around the track with initial values for P, I, and D set to 60, 0.1, and 10. The results that were received were quite amazing. The resulting P, I, and D values were found to be 75.036, 0.0552, and 7.5116. This resulted a lap time of 7.4 seconds and an error of 4132. This yields a score of 5762. This lap time is very fast in comparison with the values from the manual tuning process, but it appears it comes at a cost a very high error.

# Conclusion

In conclusion, this project demonstrated the use of a PID controller to follow a line laid out by the user. It also opened the door to all of the problems that hardware brings with it. As a lessons learned, fixing the hardware problems that arose during the creation of this robot took longer than the software problems. Although this line following robot does well, it has many areas it is lacking such as low memory capacity, low processing power and lack of threading. This kept us from being able to save off data from the longer runs which would be needed to learn PID values well and kept us from being able to transmit data while the robot was learning. There are also some areas it could improve. One improvement in particular is that the microcontroller board used came with an onboard motor driver. This motor driver was unable to carry the power needed to drive the high speed motors we intended to use on the system which require higher current than our motor drivers can provide. The motors in combination with the subpar motor controller have yielded less than ideal results. Therefore, another improvement is to get faster motors. A side effect due to this change is less battery life, but with a rechargeable battery this shouldn’t be an issue.

The learning technique outlined in this project showed signs of great improvement over the manual tuning process with respect to faster run, lower error, and most importantly with fewer manual interruptions of the learning process. This learning process is a real appealing feature for anyone who has had to pleasure of manual tuning a PID controller. There are some leading improvements that are required to have a viable learning machine. The first of which is a way to store the values of the best PID so that sequential runs can take place using the best value from the previous runs. In addition, switch should be added to allow for the robot to start in learning mode or performance mode. This would allow for the users to train on a particular track, then, switch to use the best PID values found.

# References

Collins, D.; Wyeth, G.: Cerebellar control of a line following robot. *In: Proceedings of the Australian conference on robotics and automation (ACRA ’99)*, p. 74–79. (1999)

Dupuis, J.-F., Parizeau, M.: Evolving a vision-based line-following robot controller*. In: The 3rd Canadian Conference on Computer and Robot Vision (CRV 2006)*, p. 75. IEEE Computer Society Press, Los Alamitos (2006)

Pakdaman, M.; Sanaatiyan, M.M.; Ghahroudi, M.R.: A line follower robot from design to implementation: Technical issues and problems, *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on* , vol.1, no., pp.5-9, 26-28 Feb. 2010  
doi: 10.1109/ICCAE.2010.5451881  
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5451881&isnumber=5451211

Pololu Baby Orangutan B Users Guide. Available from http://www.pololu.com/docs/pdf/0J14/baby\_orangutan\_b.pdf.

# Appendix A

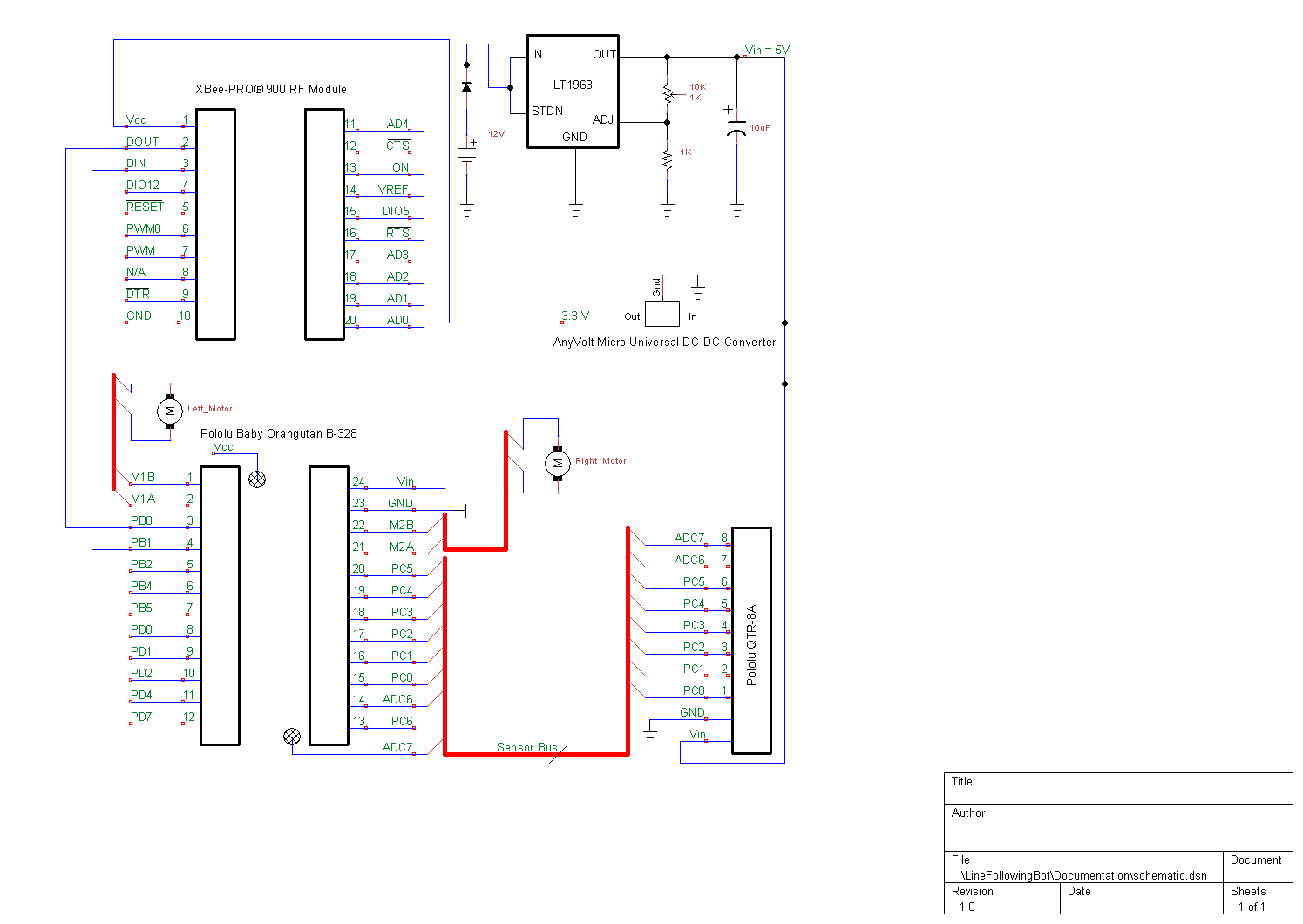


Figure – Full Schematic of Line Following Robot

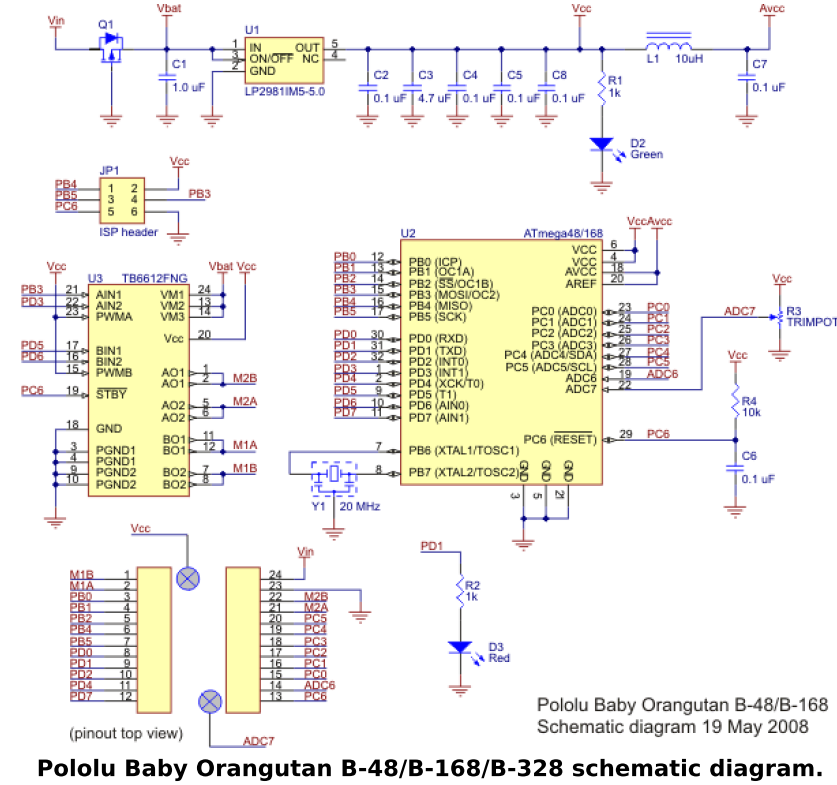


Figure – Pololu Baby Orangutan B-328 Schematic diagram (Pololu Baby Orangutan B Users Guide)