Wheel-E Self balancing robot

Team BEET
Ethan Knoll
Ben Nollan

ECE 220

Digipen Institute of Technology

16th February 2017

1 Abstract

This paper discusses our project for our sophomore year engineering class, which was a self-balancing autonomous robot. It describes the motivations, process, and result of our efforts. We also describe the sensors, hardware, and methods we used to be able to make the robot balance on its own.

2 Introduction

When we were deciding on what project to work on the most important consideration was a project that would require us to learn something useful. When we thought about creating a balancing robot we grew excited as we realized the breadth and value of the methods and sensors that a balancing robot employed and decided this would be a great project to undertake. As we researched we found that an accelerometer and a gyrometer were commonly used for this type of application and the ubiquitousness of these sensors in engineering applications really appealed to us. We also found that a control system was a necessity for our robot, and while this was perhaps beyond our capability we thought it would be worthwhile to try working with it. Another method we didn't think we would need, but ended up using was filtering of sensor data, which is another very beneficial skill. Getting to work with the techniques and sensors on our balancing robot was really rewarding, especially when we were finally able to make it balance. We believe we learned a lot and are also happy with having created a delightful and intriguing robot.

2.1 Use Model

Our robot is designed to be intuitive and simple to use. Once the robot is turned on it will not attempt to balance itself until it is held in a vertical and balanced position straight up. When the robot senses that it is in its most upright and balanced position it will begin sending power to the motors and will balance on its own. In the future we intend to connect a remote to the robot so that it can be controlled by rotating and also be able to move backwards and forwards.

3 Design

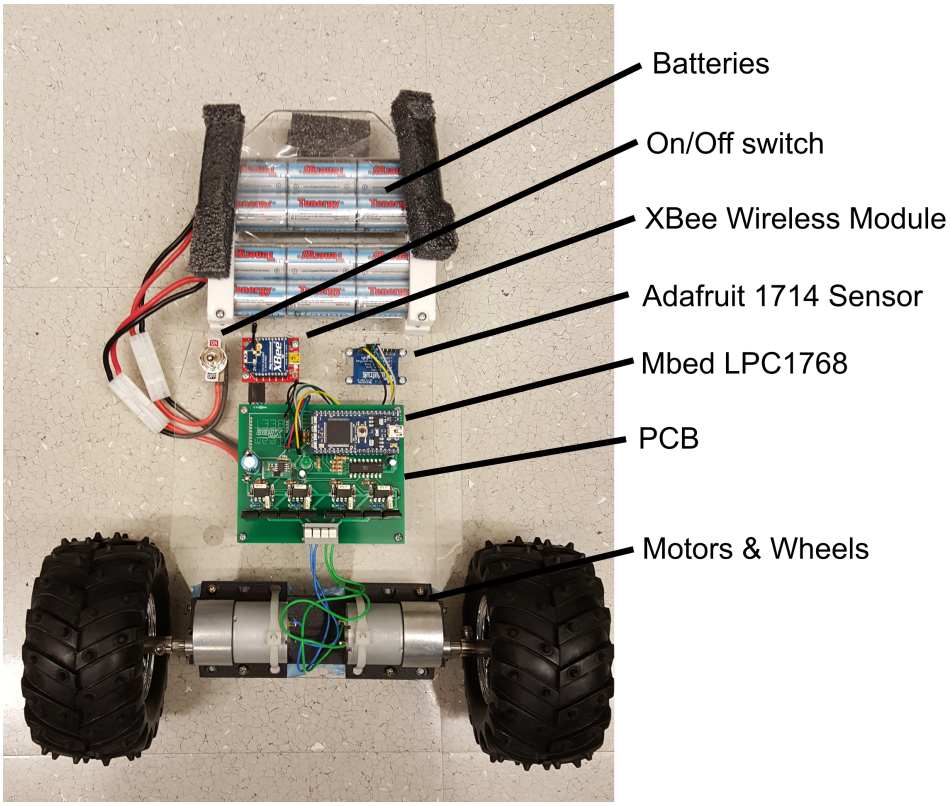


Figure 1: Labeled Robot

3.1 Mechanical

As we began to conceptualize how we could build this robot we decided we could mount all the components we needed on one board that would stand vertical as the robot balanced. We mounted the motors at the bottom and the batteries at the top to maximize stability. The Sensor board is mounted directly below the batteries so that the battery's weight acts as a damper to a lot of the vibrations caused by the motors.

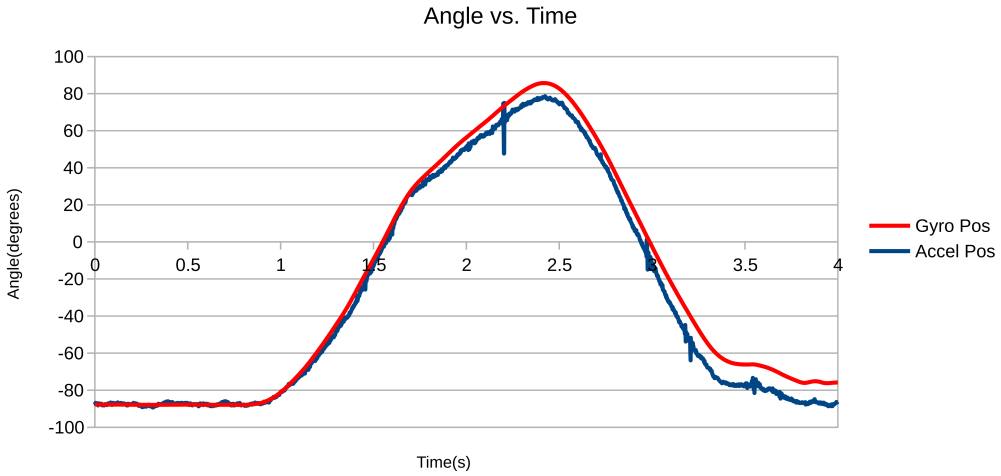


Figure 2: Comparison of angles from the robot being slowly rotated.

3.2 Sensors

We knew we needed an accelerometer and a gyrometer because examples of this type of balancing robot we found online all used these two sensors(1). After we recieved our sensors thought we thought we could get by using only the gyroscope and monitoring our output to the motors, but the drift of the gyroscope made it too inaccurate to use on its own. Then we tried using an accelerometer and a gyrometer because combined their data with a simple filter as described in another online tutorial(2) gave us a quick and accurate tilt angle of the robot. If just a gyrometer were used, the angle would drift over time.(Figure 2) If only an accelerometer was used, the angle would be very inaccurate whenever the robot was moving.(Figure 3) We chose to go with the board from Adafruit because the accelerometer and gyrometer came on a single board, were on the same data line for I2C, and the gyrometer had 16 bits of precision.

3.3 Software

On startup our robot has an initial calibration phase that determines the zero offset of the gyrometer, this is important because when the gyrometer is not moving at all it has a small offset from zero. The sensor data is

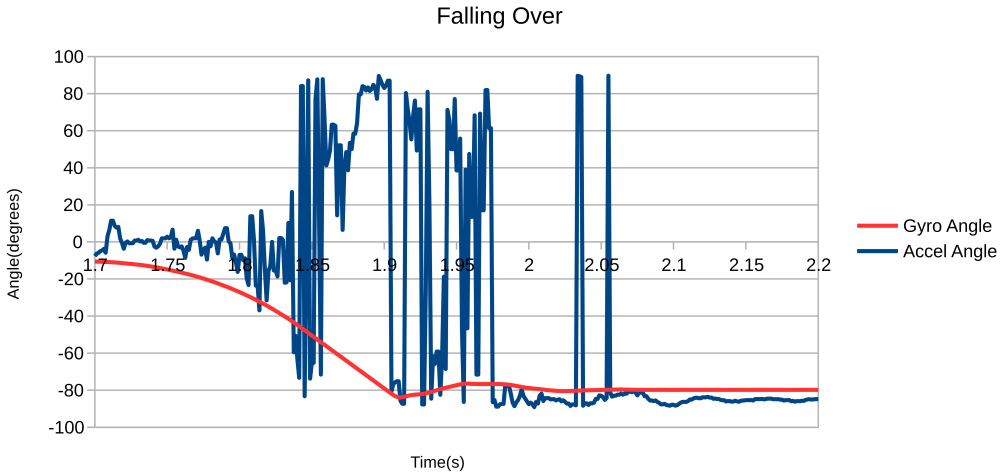


Figure 3: Comparison of angles when robot falls over.

passed through a simplified Kalman filter to combine the accelerometer and gyrometer data. The sensed angle is then fed into the feedback loop. This in our case is a PID loop, a loop

4 Parts

Part Number	Manufacturer	Value	Description	Voltage	I(mA)	Supplier	Price	Quantity
1102	Pololu	19:01	Metal Gearmotor	0-12V	3000	Pololu	29.95	2
1557	Pololu	120mm	Wheels	–	–	Pololu	14.95	2
XB24-AWI-001	Digi	Series 1	Xbee Module	3.3V	¡250	DigiPen Stock	0	2
AMC1602AR	Orient Display	16x2	Character LCD Display	5V	150	OrientLCD	6.35	1
–	Tenergy	3Ah	NiMh Battery Pack	7.2V	–	Digipen Stock	0	2
1714	Adafruit	9-DOF	IMU Breakout	5V	¡100	Adafruit	19.95	1
OKI-78SR-5	MPC	–	DC/DC Converter	5V	1500	Mouser	4.30	2
LPC1768	NXP	–	ARM mbed Microcontroller	3.3-5V	¡250	Digipen Stock	0	1
–	Digipen	–	Scrap Acrylic	–	–	Digipen Stock	0	1
–	Molex	–	RC Battery Connector	–	–	Digipen Stock	– 0	1
PIC18F4520	Microchip	–	PIC Microcontroller	5V	¡100	DigiPen Stock	0	1
NCP5106BPG	ON	–	Mosfet Gate Driver	10-20V	500	Mouser	2.38	4
2SK3703	ON	–	N-Channel Mosfet	60V	30A	Mouser	1.53	8
MIC29300	Micrel	3.3WT	3.3V Regulator	4-30v	3000	DigiPen Stock	0	1
LTV-845	Lite-On	–	Optocoupler x4	–	–	Mouser	0.88	1

5 Technical Issues

5.1 xBee

When trying to use the xBee we discovered that if the device received too much data the link between the two would not transfer any data until one of the xBee modules was power cycled. Our solution to this is to have the remote not send any data to the robot until it acknowledges that it has received the previous data that was sent. The data communication works but still has random stalls for unknown reasons.

5.2 LCD

When first trying to get the LCD to print to the screen it wouldn't work, it just wasn't getting the data. After several hours it was figured out that the PIC cannot read the state of a port, so all of the operations that only were meant to modify one pin modified the whole port. The solution to this was to write wrapper functions around each port that had a buffer built in to handle setting of individual bits.

5.3 I2C

When we first began working with the I2C protocol we weren't getting any data back from the sensors to our microcontroller. The datasheet for the sensors listed a particular sequence of data transmissions to communicate with the microcontroller and the mbed compiler offered unique functions to follow these sequences. Unfortunately this didn't work so we had to try a different method. Fortunately the mbed compiler also offered all in one functions for I2C communication like read and write, which would perform the sequence described in the datasheets of the sensors automatically. When we used these functions correctly we finally got data from our sensors.

5.4 Angle of Robot

When we first began trying to calculate the angle of the robot we calculated this by first taking the arctan of the values we received from our accelerometer from the z and y axis. This give us the correct angle, but with some noise and without the response that we would need. Then we

tried to determine the angle from the integral of the gyrometer because the gyrometer provided the rotational velocity of the robot so the integral would provide an angle. When we performed this calculation we did get a correct angle calculation and it was in fact more responsive than the accelerometer, but there was drift because the integral introduced error, which grew over time. We thought we could neglect this drift by accounting for power output to the wheels, but this didn't work. We decided we needed to merge the sensor data with some type of filter, the way other projects have described, but the kalman filter seemed too complex for us. We found a website, "http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/step2/Gyroscope/", that described a filter that could be applied to our sensors that was modeled off of a simplified kalman filter. The idea behind it was to use weighted values of the gyrometer and accelerometer data to return an angle measurement. It also would use the speed of the gyrometer to check the change in angle against the previously weighted angle we had calculated for every iteration. After we implemented our own version we found we had a very accurate angle with much less noise and without drift.

5.5 Control Loop

At first we tried to apply a control loop we found online that used only the data from the gyrometer and the power output to the wheels to gauge where the robot was and to balance the robot. We could not make this work in our implementation because the robot became unbalanced very quickly. We have since modified how we determined our angle and have a promising idea from a website kindly shown to us by our teacher about how to implement another control loop, which is our next step in our project.

6 Testing and Design Verification

6.1 Unit Tests

6.1.1 Robot

Action	Results	Troubleshooting
Volt meter across battery.	Each battery should be between 7.2 and 9 volts.	Replace/recharge battery.
Wiggle battery connections.	Robot should stay on.	Clean/replace battery connections.
With batteries connected, turn on the power switch and measure the voltage across the switch.	Should be zero volts or very close to zero.	Replace the switch.
Place a voltmeter on the output of the voltage regulator.	Should be 5volts.	Replace regulator.

6.1.2 Controller

Action	Results	Troubleshooting
Check voltage of batteries.	Should be between 0.9V and 1.6V	Replace Batteries.
Check voltage on regulator.	Should be between 7.2V and 12V on the input and 5V on the output.	Replace regulator if output not right, otherwise check the wires to the battery.
Move steering and throttle.	Values on screen should move smoothly.	Replace potentiometers.

6.2 System Tests

To access the system test menu on the controller, pull the trigger and then turn the power switch on.

Pulling the trigger on the LCD Test	Will color the entire screen black.	If dead pixels or lines are observed, replace the screen.
Pulling the trigger on the xBee test.	Will start sending packets to the robot.	If no packets are received, replace xBee.
Pulling the trigger on the sensor test.	Two bars should appear on the remote screen, one for the accelerometer angle and one for the change in angle from the gyrometer.	If the data does not look correct, replace sensor board.
Moving the throttle on the motor test.	Both motors should spin in the same direction.	Check motor wiring.

6.3 Built in Self tests

The robot monitors various components while it is running, which are represented by the LEDs on the front of the robot.

Led 1 on the microcontroller is lit.	Sensor board communication has failed.	Check wires to sensor board / replace sensor board.
Led 2 on the microcontroller is lit.	Battery Voltage is low.	Recharge batteries.

7 Conclusions and Future Work

When we first thought about building a balancing robot we really hoped it would help us learn about some important concepts and excite us about work in engineering and after seeing our robot balance we think that we have accomplished all of our goals. We learned about working with the ARM microcontroller and actuators. We learned about I2C and getting data from our accelerometer and gyrometer. And we struggled with control loops and filtering our data to get a real time measure of the tilt of our robot. Although we may have over scoped our project, having something to show for it has

really energized us to continue working on it. We're excited to make it more robust while it balances and create a remote control for it. While this project was challenging, and at times we feared it wouldn't work and had all been in vain, finally seeing our robot balance helped us appreciate how much we have learned and has inspired us to continue creating something we can really be proud of.

8 Author Contributions

Ben Nollan

- PCB
- Controller Schematic
- Motor mount
- Controller Code
- Robot Code

Ethan Knoll

- PCB
- Robot Code
- Control Loop and Filter Research
- Testing

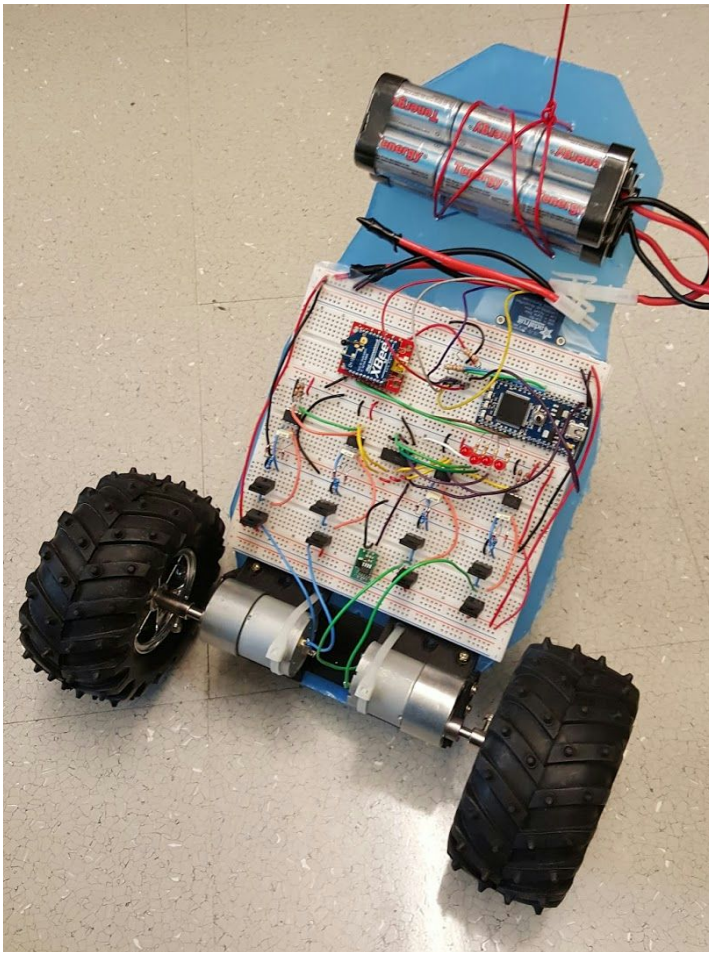
9 Bibliography

(1) Robocv.blogspot.com, 'Computer Vision, Robotics and Arduino: Making a two-wheeled self-balancing robot', 2015. [Online]. Available: http://robocv.blogspot.com/2013/0 two-wheeled-self-balancing-robot.html. [Accessed: 05- Dec- 2015].

(2) Gadget Gangster, 'Accelerometer & Gyro Tutorial', Instructables.com, 2015. [Online]. Available: http://www.instructables.com/id/Accelerometer-Gyro-Tutorial/. [Accessed: 05- Dec- 2015].

10 User Guide

Wheel-E User Manual



Team BEET
Ethan Knoll
Ben Nollan

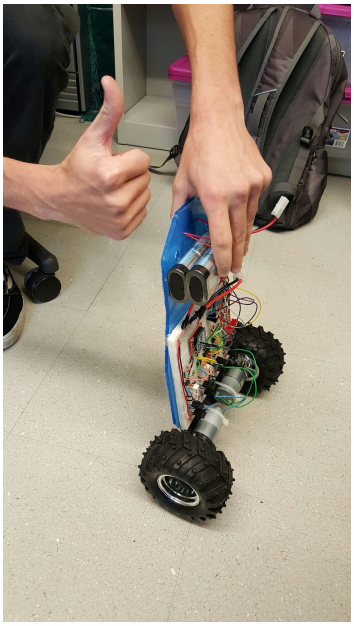
Getting to Know Your Wheel-E

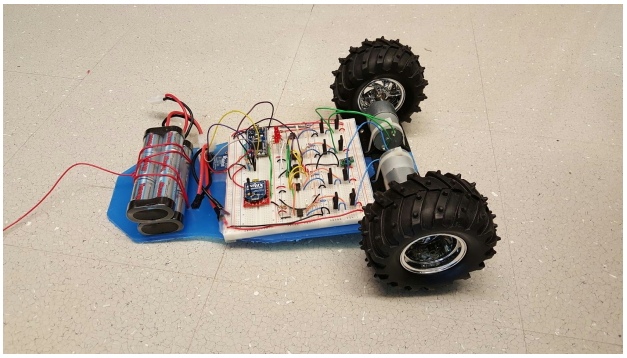
If you are reading this then you have recently acquired and are in possession of a Wheel-E autonomous self balancing robot! The Wheel-E is designed to stay upright on its own two wheels and remote controlled for hours of interactive enjoyment. While there are only a few components and setup is not required, a simple understanding of the robot will ensure robust and long term operation of your new Wheel-E!

This manual explains the basic operation and maintenance of the robot. Please read carefully and keep for future reference.

Getting Started

- How to Begin Balancing the Robot
  - Have the robot laying on a flat surface.
  - Turn the power switch on the robot to “ON”.
  - Wait for the green status light on the robot to start blinking (should be about one second).
  - Lift the robot to a balanced upright position and wait until it starts trying to balance to let go.
  - At this point the robot should maintain an upright position by itself.





- Controlling the Wheel-E
  - Turning the remote on by
    - Verify that there are fresh batteries in the remote and turn the switch on.
    - This will turn on the remote and the text “Wheel-E” should be seen along with two indicator bars.
      - The left bar indicates turn and is controlled by the wheel
      - The right bar indicates forward and backward movement and is controlled by the trigger.
    - The remote will connect to it automatically.
  - The robot can now be remotely controlled by pulling the trigger of the remote
    - To move the robot forwards pull the trigger.
    - To move the robot backwards push the trigger.
  - Use the wheel on the remote control to turn the Wheel-E left and right accordingly.
    - Turning the wheel clockwise turns the Robot right.
    - Turning the wheel counter-clockwise turns the Robot left.

Troubleshooting

If your Wheel-E is not operating as intended:

- If the remote screen is blank the remote may need new batteries
- If the robot is not balancing well try turning the robot off and then back on again thereby recalibrating the robot.
- If the remote fails to control the robot, power cycle the remote, sometimes many power cycles are necessary.

Maintenance

To keep the Wheel-E operating at peak performance some maintenance is required.

- Using fully charged batteries in the Wheel-E will ensure that the robot has enough power to maintain its balance and protect it from any damage that could occur by falling over.
  - The Wheel-E uses two 7.2Volt RC car battery packs, these can be recharged using an RC car battery charger rated for 7.2Volts.
  - Your Wheel-E will not operate if these are discharged.
- Avoid rough handling of the robot to keep the components working optimally.
- Cleaning of the motors to keep debris and dirt out of the motors may be required.
- If the robot won't balance even after checking the batteries and its initial calibration there may be a problem with the software and you should return the robot to your distributor to get the latest software.

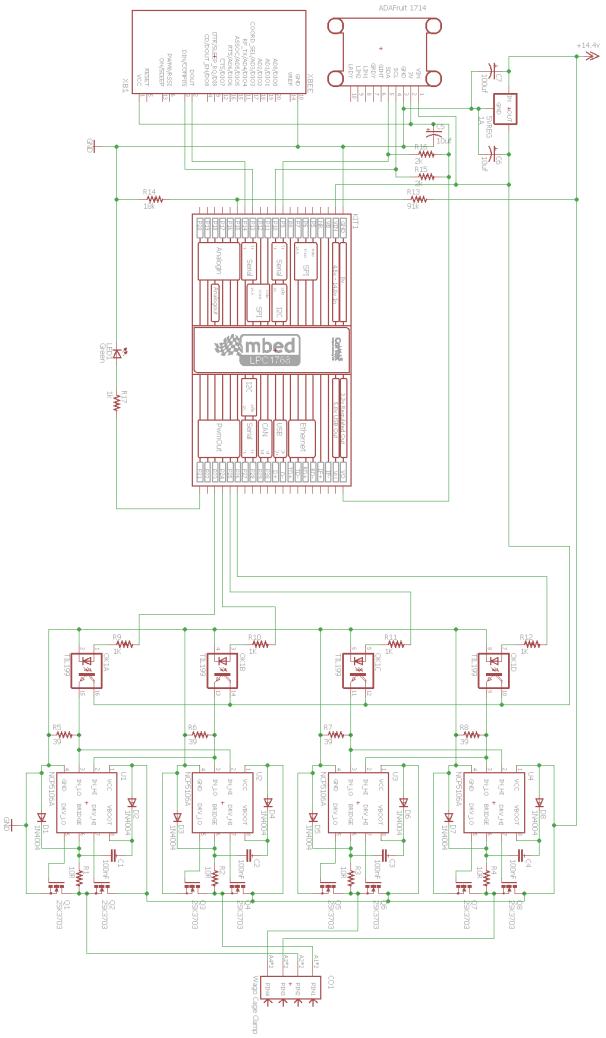


Figure 4: Schematic of the robot.

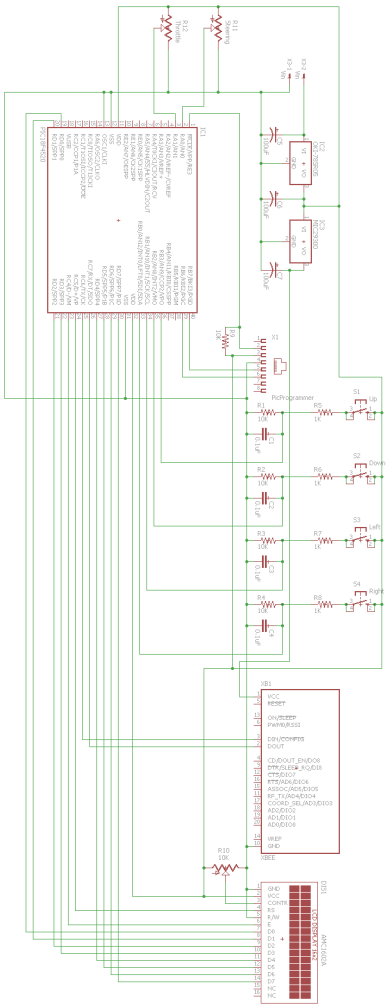


Figure 5: Schematic of the controller.

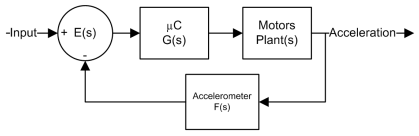


Figure 6: Flowchart of Feedback Loop

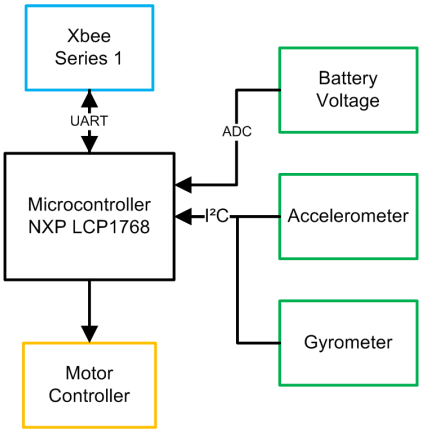


Figure 7: Block Diagram of Data on the Robot

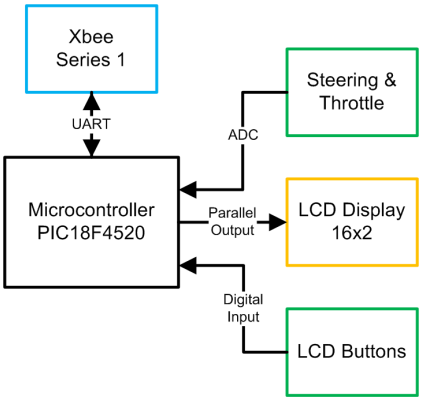


Figure 8: Block Diagram of Data on the Controller