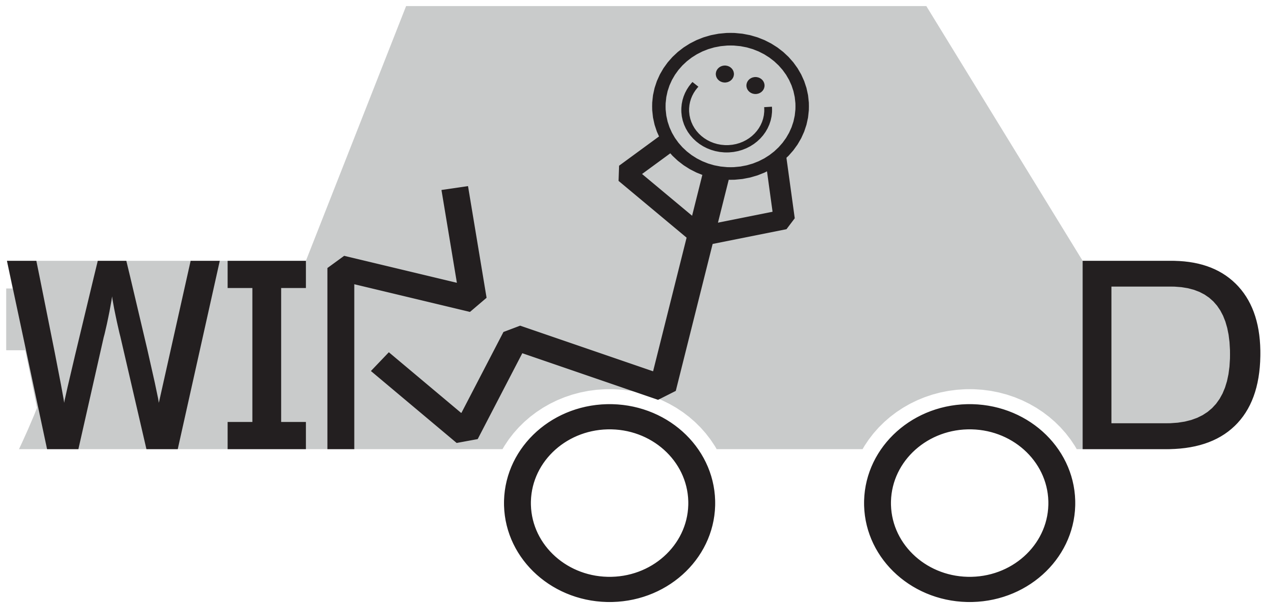


The University of Pennsylvania

ESE 350: Embedded Systems/Microcontroller Laboratory

Final Project Report - Project Winwood

Christopher Champagne & Matthew Howard

May 14th, 2015

i. Table of Contents

1. Introduction

Automotive traffic congestion leads to inefficient use of transportation infrastructure, unnecessary energy costs, increased greenhouse gas emissions, and increased travel times. These problems also lead to secondary yet significant effects including the need for additional highways and lanes, magnifying traffic congestion's environmental and economic footprint. We posit that much of this congestion can be eliminated by automating both driving and the exchange of information between vehicles sharing the road.

Most research into self-driving vehicles today focuses on systems that exist within a landscape filled with human drivers. Because these systems are bound by the conventions and rules of engagement of human drivers, they can only be slightly more efficient than human drivers in using the space of the road. Our goal for this project was to produce a set of multiple autonomous vehicles that are able to collectively make decisions in order to greatly improve travel time and utilization of infrastructure. These robots would serve as a model and demonstration of the benefits of a fully autonomous transportation environment on streets and areas with heavy traffic.

We ran into several challenges during our project, however, and we had to alter our goals for the project for this course. Our adjusted goal for the project was to have one robot be able to use the LIDAR to determine its position on the track and to navigate when given commands from a web console. We also planned to have a second robot provide limited traffic scenarios, but one of the two LIDAR units started to malfunction during the final hours of the project. Unfortunately, although we had almost all of the infrastructure in place to support multiple vehicles simultaneously, we lacked the working hardware for the final demo.

2. Bill of Materials

1. (2x) Autonomous Robot Vehicle
    a. Raspberry Pi 2 (Quad Core, 1GB DRAM) w/
        i. Ubuntu 14.04 Trusty Tahr for ARM[1]
        ii. WiringPi Library
        iii. Hiredis C Redis Client
        iv. Project Winwood Robot Control
    b. OurLink USB WiFi (802.11b/g/n) Module
    c. CV HB-401 Dual Channel H-Bridge
    d. Piccolo Laser Distance Sensor (XV-11 Neato LIDAR)
    e. (2x) #2368 Pololu 150:1 Micro Metal Gearmotor MP
    f. #1088 Pololu Wheel 32×7mm Pair - White
    g. #954 Pololu Ball Caster with 3/4" Plastic Ball
    h. Custom Chassis made from Lasercut 1/8" MDF
    i. LM7805 5V Linear Regulator
    j. 10x AA Batteries
2. 6' x 4' rectangular field with 10" walls on each side
3. Laptop for input and monitoring
    a. Redis messaging server
    b. Project Winwood Web Console Server on Node.js
    c. Web Browser
4. 802.11n LAN Router

[1] https://wiki.ubuntu.com/ARM/RaspberryPi

3. Systems Overview

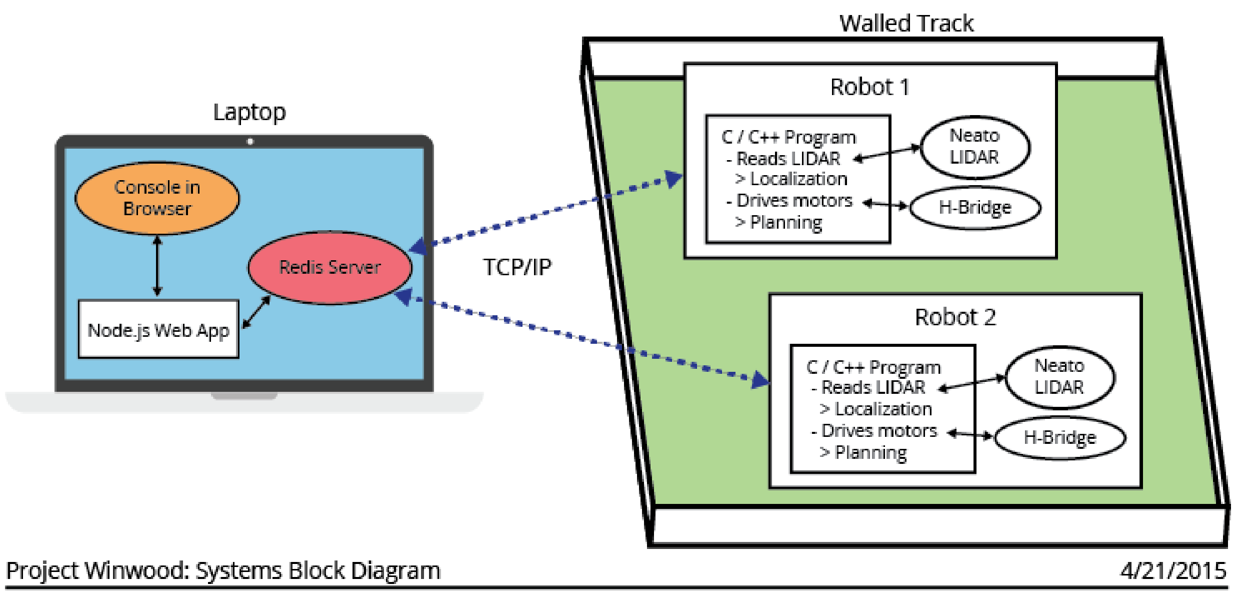


Figure 1: Systems Block Diagram

Each robot consists of a mechanical subsystem, an electrical subsystem, and a software subsystem. The mechanical subsystem consists of a drivetrain and a Neato LIDAR. The mechanical subsystem is controlled and used by the software system through the Raspberry Pi's GPIO ports and serial IO lines. The robots are each responsible for their own closed-loop control to the destination assigned to them by the laptop over Redis. The software subsystem simultaneously controls the drivetrain, communicates with the laptop over Redis, reads the LIDAR data, and processes the LIDAR data to determine its own position. The electrical subsystem provides power to each of the above components and facilitates communication between the components.

4. Descriptions of Subsystems

a. Mechanical

For our robots, we wanted them to be as small as possible while still structurally sound and able to hold all the necessary hardware while driving around. We decided that we would not be able to find a suitable chassis off the shelf so we chose to lasercut our own chassis pieces and construct the cars from scratch. We used the PLS flatbed laser cutters in the Rapid Prototyping Lab and after various iterations and adjustments created two chassis since we only had two LIDAR units. The chassis consists of two levels, a base to hold the Raspberry Pi and the H-Bridge, and an upper level to give the LIDAR a flat space with an unobstructed vantage point for reading in measurements.

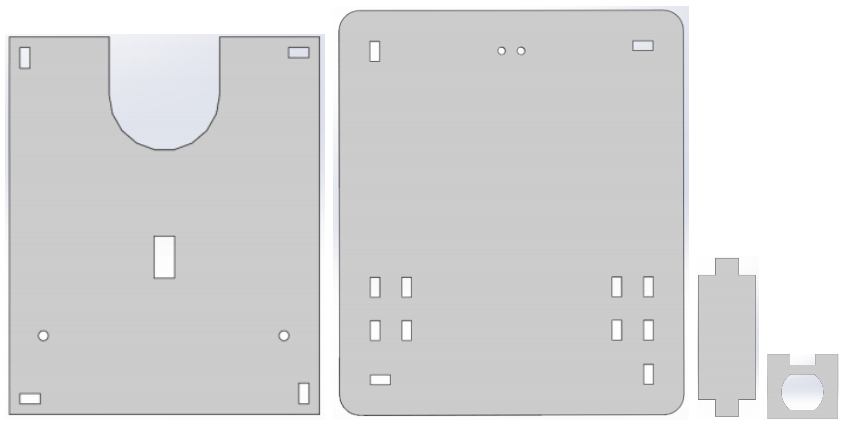


Figure 2: Design for Different Components of the Chassis
Top Level, Base Level, Custom Standoff, Motor Holder

We used SolidWorks to design the chassis. In order to ensure that the top level would be sturdy and that the LIDAR would not sway as the car moved, we had to measure and design the chassis with press fits. The cutting and building process took multiple attempts to perfect the press fit because of the laser's inconsistencies. The chassis were made of MDF (Medium Density

Fiberboard) and the assembled model can be seen below. In addition, we used a limited amount of superglue to hold the motors and press fits in place.

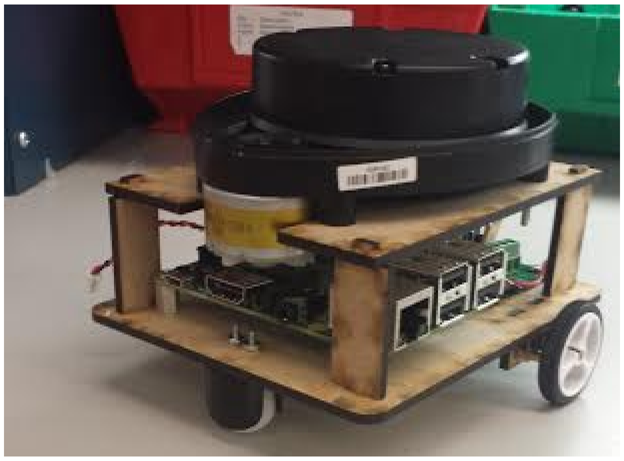


Figure 3: Assembled Chassis with Some Hardware Attached

b. Hardware

As seen in the hardware schematic below, the main components of each robot was a Raspberry Pi 2, a Dual H-Bridge, and a XV-11 Neato LIDAR unit. To power each component, we used two separate sources of AA batteries - one for the Raspberry Pi (9 volts into a 5 volt linear regulator) and the other for the wheel motors and the LIDAR motor.

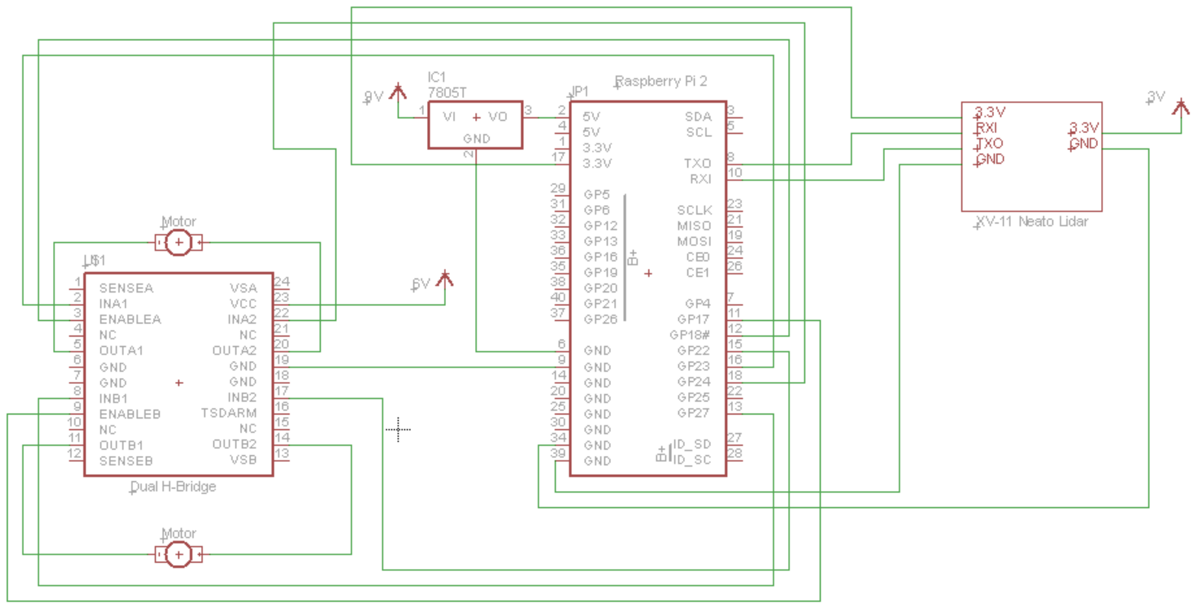


Figure 4: Robot Electrical Systems Diagram

The connections between the components can be seen above. The Raspberry Pi is powered by the output of the 5 volt regulator. It uses the TXO and RXI ports for serial communication to transmit and receive data to and from the LIDAR module, which allows the robot to determine its location. The Raspberry Pi connects to the H-Bridge and uses its GPIO pins as well as soft PWM generation to control the wheel motors' speeds (and thus the speed and direction of the robot).

Not included in the schematic above, the Raspberry Pi also connected to a Wifi Module that allowed the Pi to be programmed and allowed it to connect to the Redis server while operating.

c. Software

The software running on the Raspberry Pi was responsible for reading LIDAR data, processing that data, listening for commands from the web console, and controlling the robot's drivetrain. As a result, our program was speared into five threads, one parent thread and one for each of the four aforementioned functions. All of our robot control code was written in C, and the only non-system libraries we used were WiringPi for controlling the Raspberry Pi's IO and Hiredis for communicating over Redis.

The first thread[2] reads in serial data from the LIDAR unit. The program reads in packets of sensory information as documented in the XV11Hacking community site.[3] Then, it would write that information into a `REVOLUTION_DATA` struct, which essentially stores the LIDAR's readings as the polar coordinates of a point cloud. The `REVOLUTION_DATA` struct also holds important metadata including the number of total revolutions read in and the number of faulty readings.

The second thread[4] processes the LIDAR data. It uses the revolution count as stored in `REVOLUTION_DATA` to determine when a new reading is available. Then, it copies the data to a local array to avoid concurrency issues. It next posts the reading to Redis so that the point cloud can be graphed in the web console. The thread then performs a Hough Transform, an image analysis technique used to detect lines in a set of data. This technique is much more powerful and resistant to outliers than more common line-detection techniques such as the Least Squares Linear Regression. The Hough Transform creates a joint probability density distribution called a Hough space, which is then used to find the most likely location of the field's four walls. From the locations of the four wall relative to the robot, the program then determines the most likely location and orientation of the robot relative to the field's coordinate system. This thread is the

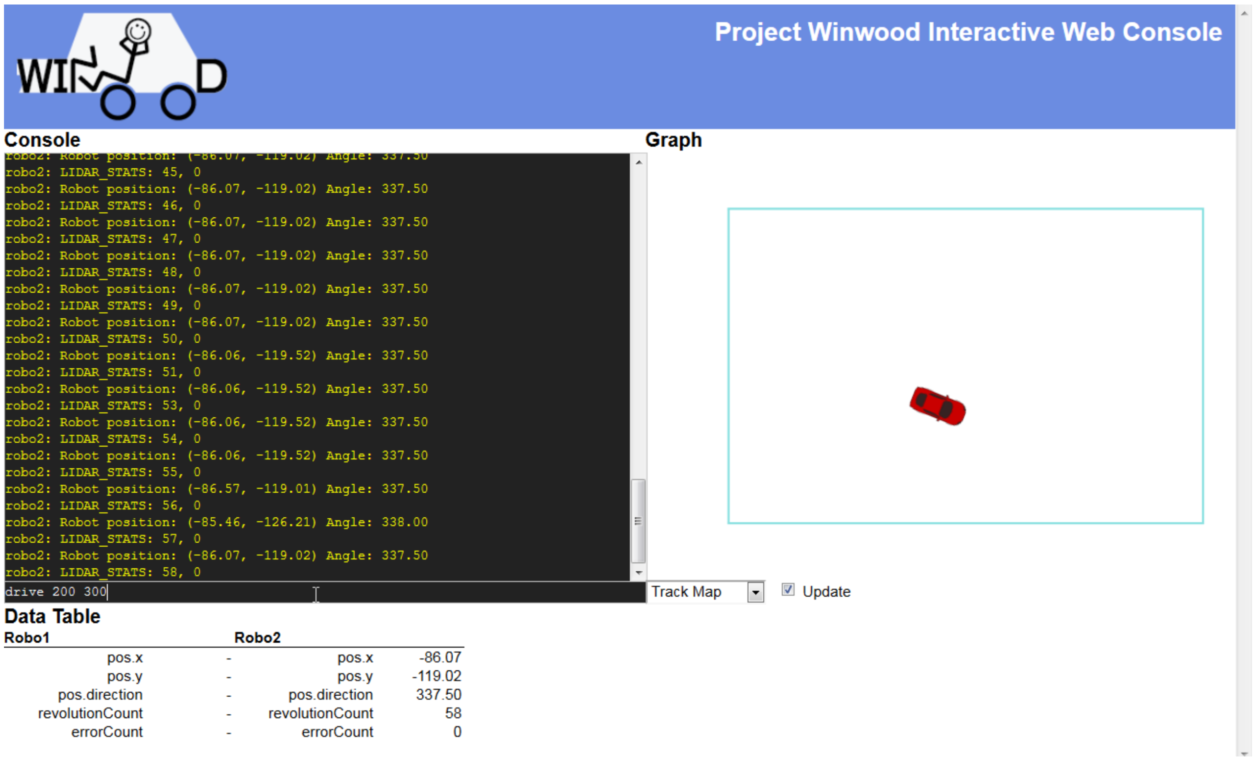
[2] https://github.com/moward/project-winwood/blob/master/LIDARTests/LIDARReadings.c
[3] https://xv11hacking.wikispaces.com/LIDAR+Sensor
[4] https://github.com/moward/project-winwood/blob/master/LIDARTests/LIDARProcessor.c

most computationally and technically involved and thus requires the power of a microcomputer such as the Pi 2 rather than that of a microprocessor such as the mBed.

The third thread[5] subscribes to a publish/subscribe channel over Redis (very similar to a topic in ROS) that allows it to instantly receive drive commands (passed as an ordered pair of coordinates) from the web console. The Hiredis library allows the program to publish to the channel asynchronously, even though subscriptions are a blocking operation in Redis. The thread shares a mutex with the driver thread to ensure there are no concurrency issues with reading and writing to the next waypoint.

The final thread[6] closes the loop by driving the robot's motors based upon location data from the LIDAR processor and waypoint data from the drive command thread. The thread will determine the direction (relative to itself) of the next point and will pivot if the next point is not within 80° in front of the robot in either direction or else will drive towards the robot on a calculated arc. The arc motion of the robot's driving favors smoothness and moving turns over the shortest distance path. The robot maintains a constant tangential speed until it is with 50mm of the target, at which point is slows down slightly. Once the robot is within 20mm of the target, it considers itself at the target and waits for further instructions. The thread sleeps for approximately 100ms after each loop iteration to avoid overcomputation, which highly increases the likelihood of errors and slows other threads such as the Lidar processor.

[5] https://github.com/moward/project-winwood/blob/master/LIDARTests/driveCommand.c
[6] https://github.com/moward/project-winwood/blob/master/LIDARTests/runRobot.c#L99



```
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 45, 0
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 46, 0
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 47, 0
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 48, 0
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 49, 0
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 50, 0
robo2: Robot position: (-86.06, -119.52) Angle: 337.50
robo2: LIDAR_STATS: 51, 0
robo2: Robot position: (-86.06, -119.52) Angle: 337.50
robo2: LIDAR_STATS: 53, 0
robo2: Robot position: (-86.06, -119.52) Angle: 337.50
robo2: LIDAR_STATS: 54, 0
robo2: Robot position: (-86.06, -119.52) Angle: 337.50
robo2: LIDAR_STATS: 55, 0
robo2: Robot position: (-86.57, -119.01) Angle: 337.50
robo2: LIDAR_STATS: 56, 0
robo2: Robot position: (-85.46, -126.21) Angle: 338.00
robo2: LIDAR_STATS: 57, 0
robo2: Robot position: (-86.07, -119.02) Angle: 337.50
robo2: LIDAR_STATS: 58, 0
drive 200 300
```

Data Table

Robo1		Robo2	
pos.x	-	pos.x	-86.07
pos.y	-	pos.y	-119.02
pos.direction	-	pos.direction	337.50
revolutionCount	-	revolutionCount	58
errorCount	-	errorCount	0

Figure 5: Interactive Web Console

The web console[7] is a lightweight Node.js application which provides a powerful graphical user interface to the Redis key-value store. It serves a web application over HTTP on port 3000 and asynchronously transfers information to the browser using both AJAX requests and web socket messages (provided by Socket.io). On the client-side, javascript can create a visualization of either the robots' location and orientation on the field (shown above as "Track Map") or display the LIDAR point cloud from one of the robots using the SVG.js[8] library. The console could be easily extended to display additional diagnostic information as well as pass additional commands to the robot over Redis.

[7] https://github.com/moward/project-winwood/tree/master/node-console
[8] http://svgjs.com/

5. Conclusion

From completing this project, we learned a great deal about power management, closed loop control, and communication between multiple systems. The project involved many mechanical and electrical components, forcing us to learn various skills, such as how to lasercut. The project also required many complex software components such as processing the LIDAR data, using a transformation to stabilize the map each robot sees, and having the system respond and drive to a designated location. Each component of the project was difficult alone, but combining them all was an entirely other problem. We had to learn how to use threading and mutexes in C to allow the Raspberry Pi to complete each task concurrently.

By the end of the class, we reached our primary goal of creating a vehicle to be able to use the LIDAR to determine its position on the track and to navigate when given commands from a user console. We also had the vehicle communicate its position to the main console, which would allow other vehicles to know its position in the future when we add more to the environment.

a. Future Improvements

1. Fix broken LIDAR and construct more cars to operate at the same time in the same world.
2. Procure Lithium Polymer batteries for the Raspberry Pi so that the robot can operate longer and is lighter.
3. Extend our code to build a virtual map and road structure for the robots to drive along using our existing data structures.
4. Implement the driving communication and negotiation logic which designed.

5. Fine tune the closed loop control so that the cars can drive to their destination smoother.

After these improvements are implemented, these robots can be used to model traffic situations and can reap the benefits of a fully automated transportation infrastructure. This is in the hope of showing how a similar system can improve transportation infrastructure utilization, especially in cities and other crowded roadways. While not all of our goals were reached, we viewed this project as a success because it is a solid base for implementing the driving negotiations (we built in structs waiting to be used for maps). Since we built a lot of our system (driving and data processing code, chassis, and communication protocol) from scratch, the design decisions that we made along the way suits our project to be extended to implement these future improvements.

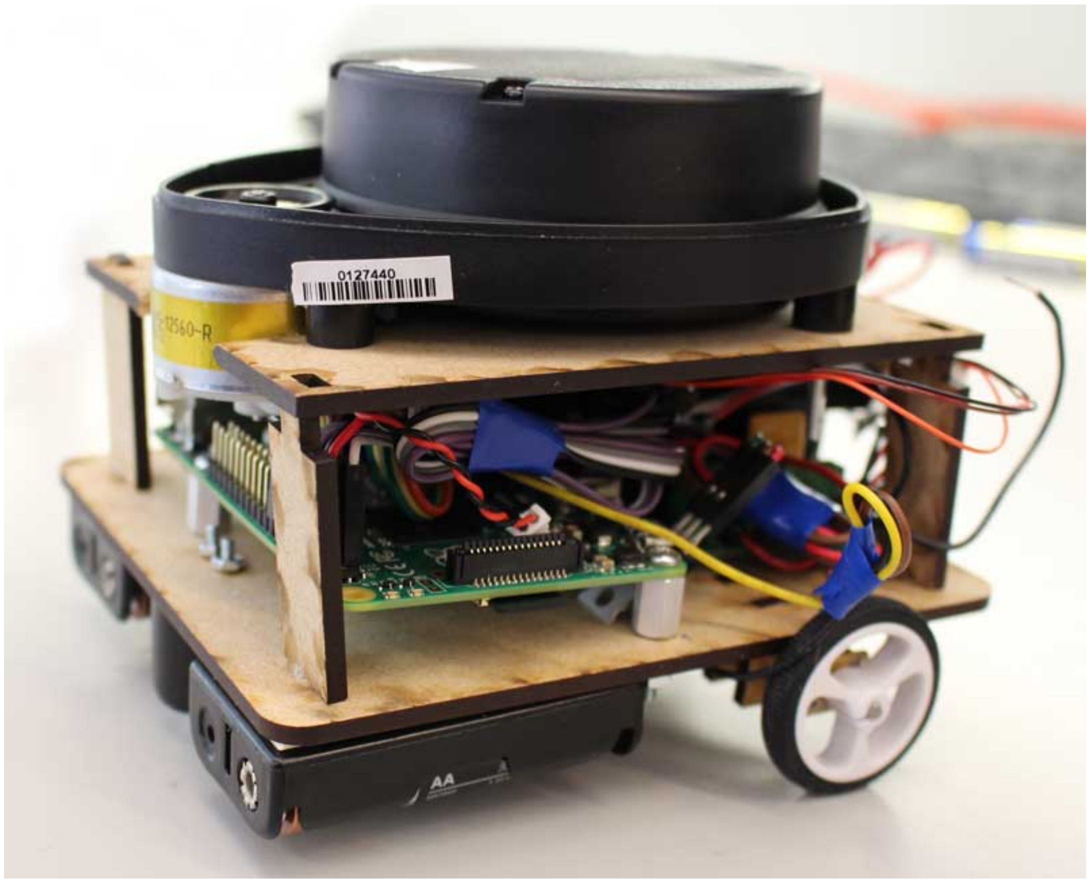


Figure 6: One of the finished robots

6. More Information

- Project Blog - http://winwood.matt.how
- Project Git Repository - http://github.com/moward/project-winwood
- Final Demo Video - https://www.youtube.com/watch?v=n0VZiIdv8so
- ESE 350 Course Webpage - http://www.seas.upenn.edu/~ese350/