10 May 2017

Auto-bot
Team 14

Final Design Report

Josiah Markvluwer (Mechanical)

Levi Dobson (Electrical)

Peter Jung (Electrical)

Peter Ye (Electrical)

Calvin College ENGR339/340 Senior Design Project

Abstract

Project Auto-bot is a senior design project done at Calvin College by Peter Ye, Peter Jung, Josiah Markvluwer, and Levi Dobson. The goal of the capstone is to combine experience earned while at Calvin College and leverage it into a year-long engineering design project to satisfy the requirements of ENGR339/340. The goal of Project Auto-bot is to create a rover that can navigate a user-given route while detecting obstacles and avoiding collisions. The project began at Calvin College in the fall of 2016 and was concluded May 6, 2017. The intended purpose of the project was to showcase the advancements being made in autonomous robotics on a small scale and prove that the technology of rover automation is available to be used and achievable at non-industrial scales. Through research of different techniques, the team built a rover that could navigate itself and safely drive a course with obstacle avoidance techniques. The team is hopeful that future teams may modify the cart design with a specific purpose in mind and continue to make the system more robust and available to handle non-ideal scenarios. The limitations facing the final design include the fact that the system relies on a good GPS signal to navigate and the rover handling protocol of objects coming from behind the rover is not robust enough to handle all situations that it could be faced with. The rover uses a combination of sensors and a waypoint controller to generate decisions enacted by a speed controller using Pulse Width Modulation signals to the motors.

Table of Contents

1 Table of Figures

2 List of Tables

3 Introduction

The goal of this project was to create a vehicle that utilizes distance and camera sensors to follow waypoints assigned by a waypoint module that uses positional systems in order to determine its location. The team made a prototype of an autonomous vehicle which can drive around Calvin's campus. This was an important problem because autonomous vehicles are typically thought of as expensive systems which can only be developed in high technology industrial settings. The goal is to realize a small-scale system with the same concepts that self-driving cars utilize today.

With these goals in mind, the final deliverables for the project consisted of a mechanical car and a motor system that uses sensor inputs in order to provide appropriate feedback to the car. The nature of this project is probably one that could be expanded upon by future senior design teams at Calvin College since many features could be added to the vehicle system if time were more permitting. Since vehicle autonomy is something that is becoming the future of the car market, it is extremely relevant to all electrical and mechanical engineers graduating in the 2010s. This project then not only serves as a great introduction to the field of automation but also is a good test pilot for what driving automation can look like in small scale settings.

The team consisted of three electrical engineering students, Peter Ye, Levi Dobson, and Peter Jung; and one mechanical engineering student, Josiah Markvluwer. This project was part of a year-long senior design course, which was required for every senior engineering student at Calvin College. The goal of this senior design course was to give students opportunities to work on real-world problems that suit their passions.

4 Project Management

4.1 Team Organization

The roles of the team members are diverse and yet all collectively working together. Team members were individually tasked with certain aspects of the project but at the same time collectively worked and supporting one another in finishing the project and completing the final outcome. The team members' specific tasks were as follows. Peter Jung developed the vision system with cameras and LIDAR software work. Levi Dobson worked on the LIDAR unit development and also Differential Global Positioning System (DGPS). Peter Ye did the motor control interface including the Pixhawk Waypoint controller and the Arduino microcontroller Pulse Width Modulation signal generation. Lastly, Josiah did the vehicle fabrication and kept track of meeting minutes. Mr. Eric Walstra was the team's Industrial Mentor who had experience with the self-driving vehicles industry. Professor Michmerhuizen is the team's advisor as well as a Course Instructor for Senior Design. There is a designated folder in Google drive where all documents are kept so that all team members can access critical files at anytime and anywhere there is an internet connection. Figure 1 shows the Team 14 layout.

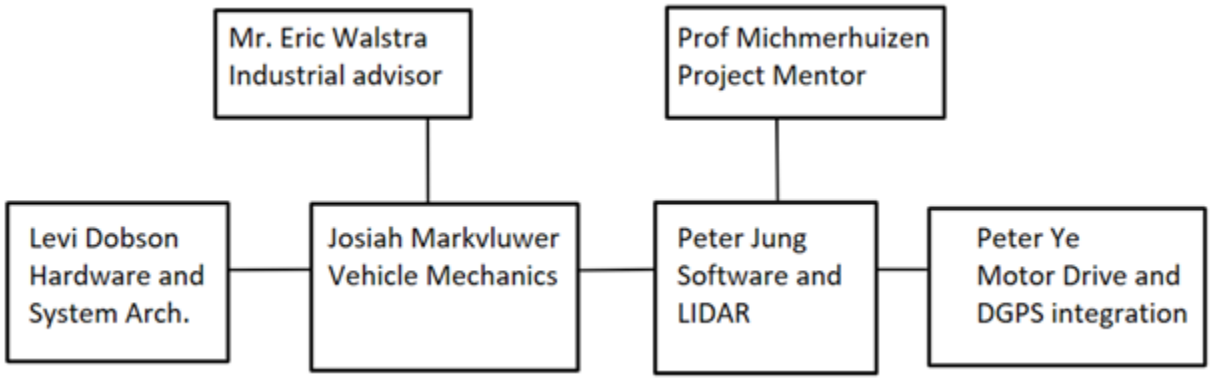

Figure 1: Team Structure

4.2 Schedule

The team's approach to scheduling first semester was to have three update meetings per week that follow the meeting time for the team's senior design class at 3:30 P.M. on Mondays, Wednesdays, and Fridays. The intent of the meetings was to go over the organization chart. Also to have a status update on timing along with how each project component was going and if the project component will meet the deadline goals made in the Gantt Schedule. The status meeting was also a time for the team to gather input from other team members. This created an environment where the team can review how realistic time estimates are for a given task. That means that the team was constantly updating and placing team members in supporting roles in order to drive through important tasks that are critical to the development of the project as a whole. Another important aspect of the status meeting was assigning a time commitment to each project and the components with the project. Josiah Markvluwer was put in charge of maintaining the schedule and keeping track of the meeting minutes and moderating the meetings to keep them on track. Josiah Markvluwer had the duty of taking notes of any critical decisions made during the meetings. These notes are stored in the team's Google Drive folders for reference. A detailed Gantt Chart is included in the Appendix.

4.3 Budget

Additional funding for the components of this project was generously given by Delphi through communication with Calvin's Regional Gift Officer Bill Haverkamp. The budget was maintained by Peter Jung, who tracked how much was on an excel spreadsheet and kept the budget up to date whenever a new cost was invoiced. The budget was referenced as an action item during the status meetings to actively track the activity of how much was available in relation to how much the team thought it might need to finish the project. Action items for when budget issues arise went to a collective decision of the team. No spending happened without the whole team's agreement and approval on an item.

The team got $800 in funding from a donation on behalf of Delphi Automotive and have used about $700 on a DGPS module from U-blox, A Pixhawk waypoint navigation system, and Arduino components.

Budget link here

4.4 Method of Approach

The approach for design was to have team members oversee areas of design and development. The team will have each member own the process and consult the team for design reviews. A big part of the design process and review is going to be seeking outside experience and resources to make ensure the design will be successful. The design lead will be in charge of calling the appropriate members to the meeting and having an organized approach throughout the meeting. The leader of each system component is in charge of the organization of workflow and following an engineering decision matrix to ensure that all options are covered. Each Leader is responsible for notifying the team when help is needed so another member can assist in meeting deadlines and requirements.

One important Biblical principle to keep in mind when the meeting is to have respect for one another and to listen before we speak. A large part of having effective design review meeting is hearing out other's ideas even when they might contradict one's own. Doing unto others as you would do to yourself is a timeless lesson that is especially important to keep in mind when working on this design project.

5 Project Justification

5.1 Proof of concept

The purpose that the team had in choosing this project was an investment in developing technologies. The reasoning is that Team 14 could display the capability of the technology available to those operating on a relatively low budget for rover projects. The concept proven by the completion of this project is that a low-cost rover can be made to direct itself while using techniques to avoid any collisions and routing itself on the user-given course.

5.2 Considered applications

This project was inspired in part by team research of other unmanned systems such as robotic automated vacuum cleaners. The initial ideas the team considered were an automated package delivery system, an automated lawn mower, or an elder care assistance robot. The elder care assistance robot application would not be applicable since Team 14 built a relatively large robot that is not made to navigate indoors.

Both the lawn mower and the package delivery systems are applications that could be feasibly accomplished with this project. The rover base is about the size of a push mower and could be used, for example, out in the country to mow a field automatically. The package delivery system application applied to this rover could work to save time by making deliveries from building to building on a local campus.

5.3 Further development

The project was made only to prove the concept that an automated rover can be made. It is possible that, in future years, other senior design teams could use this rover and modify it so that it may accomplish a task. For example, another team could use the electronics and attach them onto a mower and create an automatic lawn mower or modify the current frame in order to accommodate package delivery. Another possible use of the Auto-bot is for video surveillance around campus, which was displayed by the live stream camera installed on the rover. The hope of this project is to create a starting point for rover automation projects at Calvin.

6 Requirements

The requirements of the project are broken down into relatable parts of the design as a whole. The requirements include general requirements, hardware requirements, and Senior design requirements. The requirements are based upon the autonomous cars that are currently being tested in the streets today.

6.1 System Requirements

It shall be able to go around the Calvin Loop without any accidents and stop at stop signs without crashing into other incoming traffic. The vehicle shall be able to follow the waypoints given to it through GPS and avoid obstacles along the road and stop or slow down when needed.

6.1.1 Generic

Since the project goal is to make a scale-down autonomous vehicle, Team 14 does not have specific customers in mind, because the main purpose of the project is a proof of concept. Showing everyone the possibility that an autonomous system could be made with limited budget and time which in turn would also

6.1.2 Safety

As team 14 considers safety and harmony as an important aspect of our project, many safety features would be featured on the vehicle. First, the vehicle would have flags, warning strips, and LED lights so that the vehicle can be seen from far away and warn anyone that might encounter the vehicle. Second, the vehicle would have bumpers so that even if there is an accidental collision, the bumpers will reduce the damage done to the vehicle. Lastly, the vehicle would have manual modes and safety switches so that the team may stop the vehicle if something unforeseen circumstances occur that might damage the vehicle or hurt someone.

6.1.3 User-Friendly

The interface should make available to anyone to make a waypoint for the Auto-bot and set the starting and stopping point. It should also be able to monitor the process and make adjustments to it when something goes wrong.

6.2 Hardware Requirements

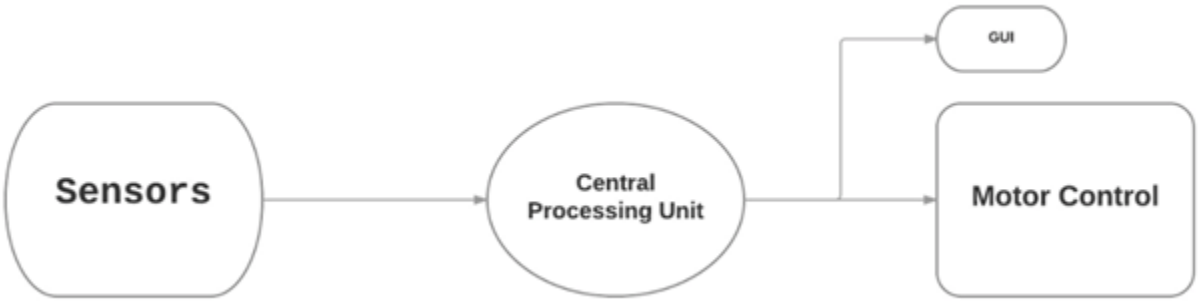

Figure 2: Main Control Components

Four parts of hardware have to be implemented for the Auto-bot to drive on the road without the assistance of human intervention (beyond inputting a destination). The Auto-bot would have to collect data from the GPS sensors and distance sensors in order to communicate with the motor control unit through a central processing unit to actuate the vehicle movement. A user interface is needed to set the starting point and ending point of the travel route.

6.2.1 Central Management

The central processing unit for the vehicle shall be small so that it could fit on the vehicle but also powerful enough so that it can process data from the different sensory inputs. It shall also be able to display the status of the vehicle and should be user-friendly so that modifications can be done easily.

6.2.2 Sensors

The sensors will have to be able to detect objects around the Auto-bot so that it can avoid the obstacles or stop if needed. Since the Auto-bot has to have time to slow down and stop, the range in these sensors would have to be between 0.5 to 5 meters. Another visual sensor would have to be implemented to detect stop signs along the road, which would have a range of 4 meters. Lastly, the sensors should be able to process the data fast enough so that the vehicle can reduce the latency and can output more accurate results giving the vehicle smoother turns and accelerations.

6.2.3 Display Interfaces

The touchscreen display on the vehicle would show the programs running on a small, but powerful computer such as a Raspberry Pi. The display would show data from the different sensors such as LIDAR and camera, which are already processed through different algorithms. The display would also be user-friendly so that simple modifications to the software may easily be done.

6.2.4 Motor and Speed Requirements

Electrical motors were used in order for the Auto-bot to drive on the road. Voltage regulators and speed controllers were needed to slow down and accelerate the Auto-bot according to the information given from the central processing unit. The car should be able to go up to a maximum speed of 16 km/hr. and have a stopping distance of about 1 meter.

6.2.5 Server and Live Stream

The team did have a live video stream on our website that looked great in our senior design presentation day. The website would also include data stream from the LIDAR, but it did not work out since we did not have enough time. This would have been great to showcase the vehicle stopping to an obstacle and clearly demonstrate how the vehicle can autonomously drive around campus without crashing into obstacles.

6.3 Class Requirements

The Senior design course requires that each group has a complete project proposal feasibility study (PPFS) before December 12, 2016. The final report and a working prototype will be done before May 6, 2017. Throughout the year, presentations and the project website will be updated to show progress. There would also be weekly meetings where team members would combine their individual works and test the final prototype.

7 Task Specifications and Schedule

7.1 Organization of Tasks

Tasks were organized by group decisions at the team meetings. Each person who was assigned a task was also assigned a deadline. While not every deadline was met it was the task owner's responsibility to communicate the task status back to the team. Tools were also used to keep the team on track such as Asana and a workflow diagram (Figure 3: ***Appendix B***)

7.1.1 Fall Semester: PPFS

The PPFS was a collective project that was split up by section for each group member for individual work. The report was then revised and read through by the group collectively.

7.1.2 Spring Semester: Final Report

The Final Report was a collective project that was split up by section for each group member for individual work. The report was then revised and read through by the group collectively.

7.1.3 Fall Semester: Research

The Final Report was a collective project that was split up by section for each group member for individual work. The report was then revised and read through by the group collectively. The research done was mostly searching for components that suit the project best. For LIDAR the obvious solution was to use the unit that was donated as the specifications well exceeded the project's needs. For waypoint navigation, the decision was made to choose Pixhawk based on the intuitive software and optimal price point. Last, the DGPS was chosen for its accuracy and ability to be within the budget of the vehicle. Decision matrices were made for each to evaluate the needs and capabilities of each component

7.1.4 Fall Semester: Oral Presentations

Josiah made the initial introductory presentation for the group. This presentation was focused on showcasing the project to students and professors giving them a high-level outline of what the team was setting out to accomplish. This included a project description, specification, and faith perspective.

Peter Ye and Levi presented a phase two update showing the more in depth component analysis and integration of parts. The presentation also showcased the feasibility of the project and what how Christian

design norms of stewardship and transparency related to the project.

7.1.5 Spring Semester: Oral Presentations

Josiah and Peter Jung gave the update presentation on obstacles and accomplishments for the group. The team focused on latency issues with the camera and vibrations in the wheels for what was overcome. The other primary focus was to show what testing and considerations had been put into place.

The team collectively made the CEAC presentation, a project showcase to the Calvin engineering board. This presentation was designed to showcase the project and explain the different components as well as how they all worked together to allow the car to drive itself. In this presentation, the team was given constructive criticism and shared concerns by the board.

7.1.6 Website Development

A website was made using WordPress to show the project's development. The link to this site can be found http://engr.calvinblogs.org/16-17/srdesign14/. The website includes a "Home" section describing the project, a "Documents" tab showing the documents related to the project, and an "About Us" that describes the team members. Levi Dobson was tasked with putting the content onto the site and Peter Jung also included a feed from the camera on the Rover that can be seen on the site. Additions were made incrementally throughout the year.

7.2 Summary of Tasks and progress

7.2.1 Expected Hours by Month

The expected hours per month were broken down based on the following chart in Figure 4.

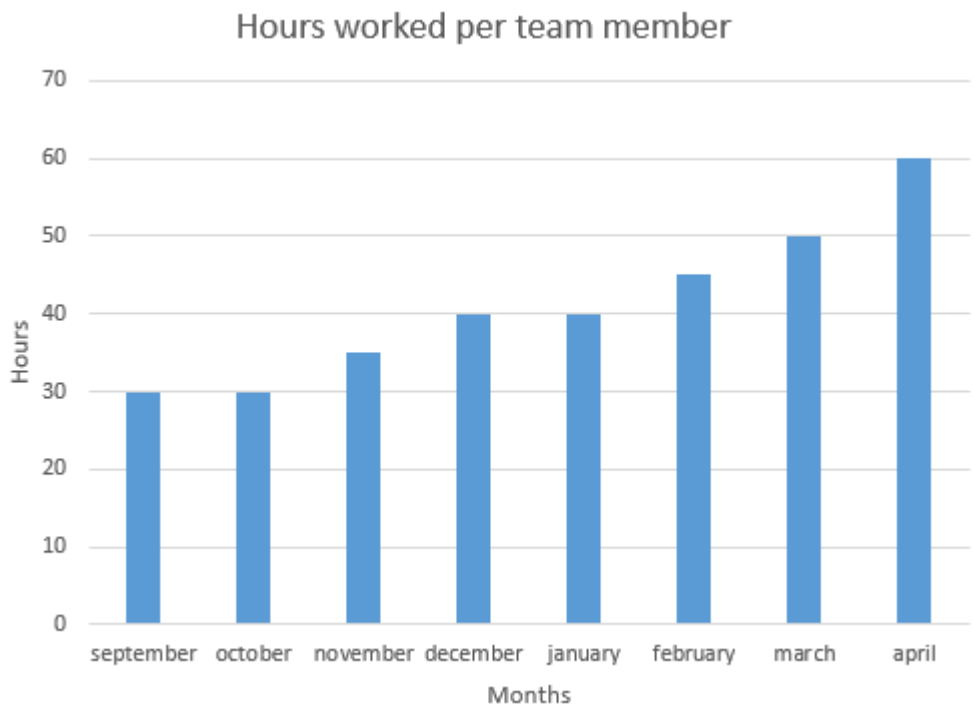

Figure 4: Time Spent per Month by Team Member

8 System Architecture

The Auto-bot vehicle system architecture has four major components that Team 14 has divided the system into. These pieces are the sensor network system, the location and navigation system, the motor drive and actuator system, and the mechanical motor and chassis system. This top level view shows that the sensor network and the location and navigation system both feed information to the motor drive and actuator. The motor drive and actuator system send signals to motors in order to drive the system. This top level perspective shows each unit in terms of its goals. Before discussing in detail the system as a whole, the components and their functions and communication methods are discussed in the next few sections. Section 8.4, interconnections, attempts to provide a more in-depth look that combines details from 8.1, 8.2, and 8.3. For simplicity, the team has divided the explanation of components into hardware and software.

8.1 Hardware

8.1.1 DGPS

Team 14 has selected the C94-M8P application board as a means of positioning. This product from U-blox is made specifically to provide “high accuracy solutions for RTK (real-time kinematics) for professional prototyping.” This is accomplished with Base and Rover station functionality, which U-blox explains with the diagram below. The base station remains stationary at a position with known GPS coordinate. It communicates with the rover station which is put on the rover’s frame and helps the rover station correct its drafting signals from satellites to achieve 2.5 cm accuracy.

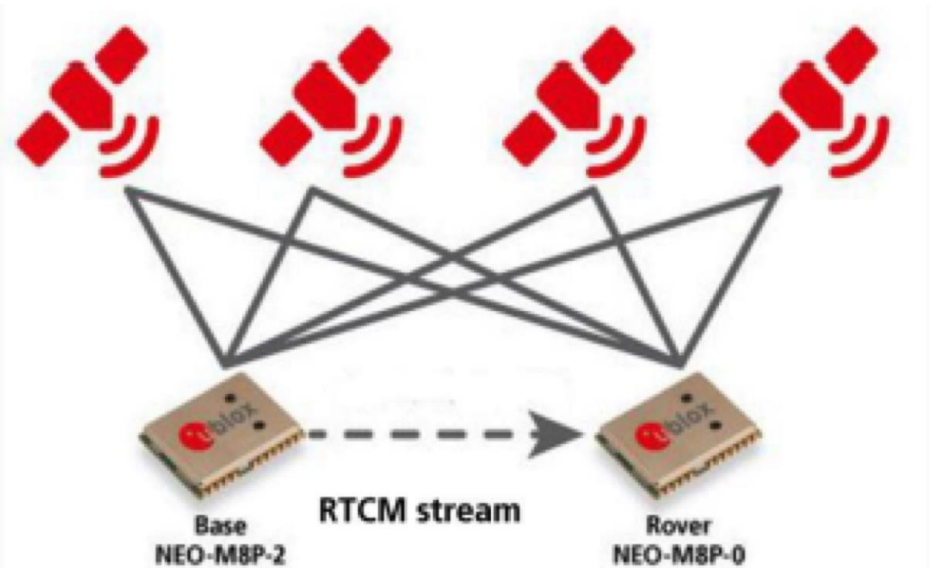

Figure 5: DGPS Diagram

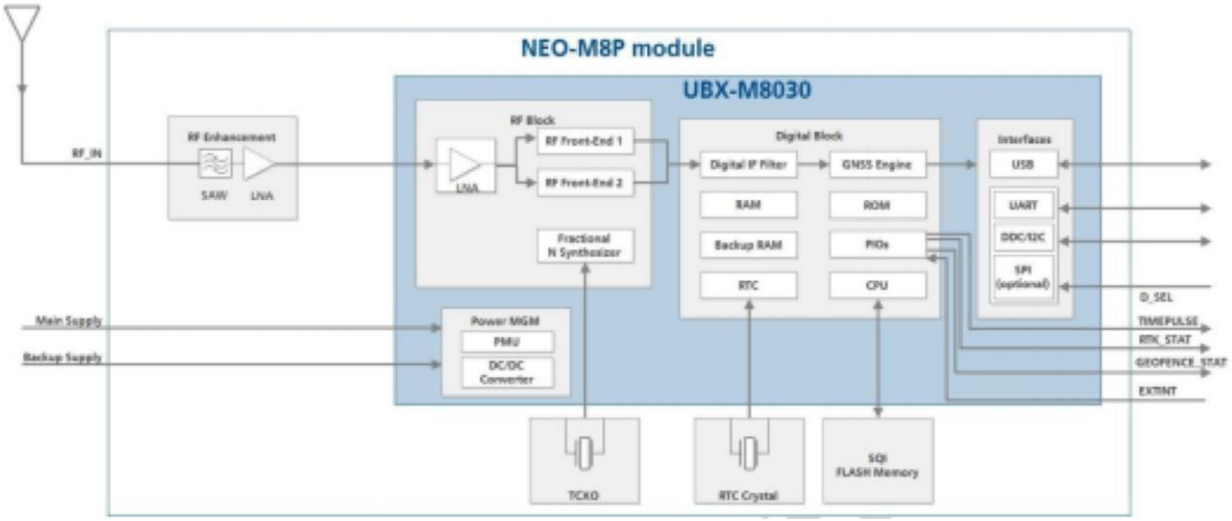

Figure 6: NEO-M8P Module Diagram

This product comes with two NEO-M8P chips. The total package contents of this system are shown in the table below.

Table 1: DGPS Unit C94-M8P – Hardware Pieces

· **Hardware Content**	· **Purpose**
· 2 application boards (both with NEO- M8P- 2)	· Contains the chip that will gather the NMEA standard GPS information, Contains the ports needed to communicate with the
· 2 external UHF antennas	· Communicates with the satellites to give location data
2 external active GNSS antennas	Communicates between the "Rover" station and the "Base" station to reduce error in positioning
2 antenna ground planes	Used for conducting communication
· 2 micro- USB cables	· Used for power and to transfer GNSS (global navigation satellite system) data

In addition to the necessary things to collect data, team Auto-bot is provided with many ways to interface the device as shown in the table below.

Table 2: DGPS Unit C94-M8P - Interfaces

· **Interface**	· **Purpose**
RS232	Contains the chip that will gather the NMEA standard GPS information, Contains the ports needed to communicate with the
USB	Communicates with the satellites to give location data
UART	Communicates between the “Rover” station and the “Base” station to reduce error in positioning
Antennas 1	Used for conducting communication
2 micro- USB cables	Used for power and to transfer GNSS (global navigation satellite system) data

The team received the DGPS at around December 22nd. This meant that the knowledge of this unit came from studying the manufacturer sheets and the team devoted time over the semester break and interim to installing and testing the unit. This was reflected in the Gantt chart in Appendix. The hardware was well documented and Team 14 had a good idea of the functions it would need from the DGPS unit and could expect to implement with certainty. These included implementing what U-blox referred to as its patented RTK, real-time kinematic, technology in order to reveal highly accurate data in the form of NMEA coordinates.

8.1.2 LIDAR

The second part of the system is the sensor network. The LIDAR was a great tool that was gaining increasing relevance in the field of autonomous vehicle driving. The basic operational diagram of a LIDAR unit is shown in Figure 7 below.

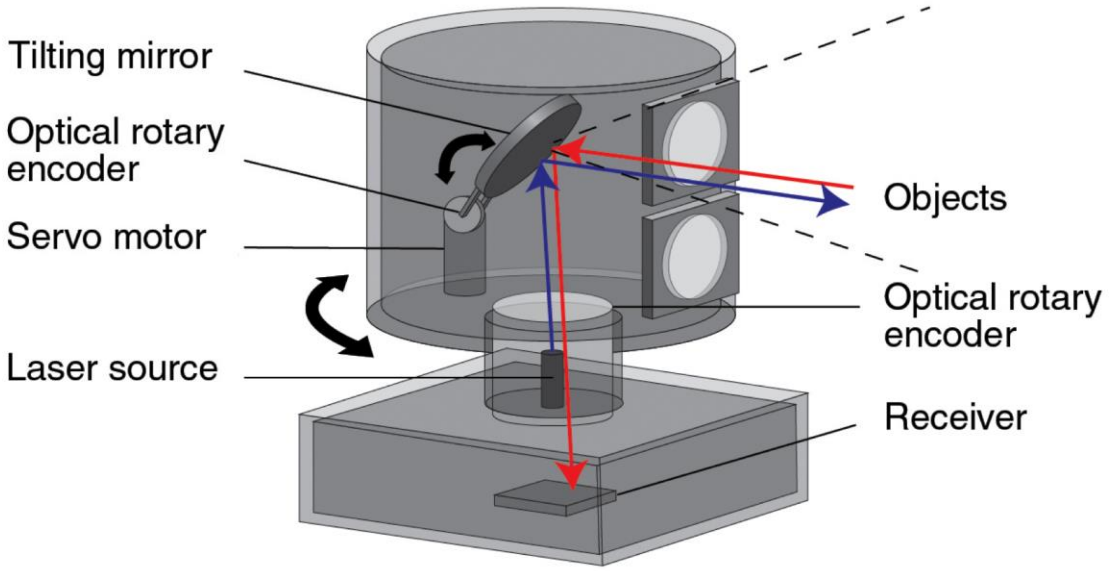

Figure 7: LIDAR Operation Diagram[1]

The LIDAR unit the team used was donated by Mr. Paul Vander Kuyl. It is a 5.5 Hz to 10 Hz, 360 degrees, and two-dimensional Robo-Peak LIDAR. It uses a serial communication with a baud rate of 115200 and has a sampling frequency of 2 kHz with a LIDAR range up to 8 meters. The angular range is about 0-360 degrees with a clockwise rotation and the resolution is about 0.5mm. The rotational frequency of the scanner can be easily adjusted using PWM to the motor. The 8-meter range gives us a lot of time to stop the vehicle in the event that an object is seen by the laser scanner in front of the vehicle.

8.1.3 Motor Drive Controllers

The current motor drive system uses Victory SP speed controllers from last year's RoboSnow design team. Team 14 has decided to use an Arduino PWM driver board that will take inputs from Arduino Uno board and use code to send a command to the speed controller that will run the motors. The library used for the PWM driver board can be found at [2]. The speed controller is shown below along with the Arduino as it is hooked up and running the motors.

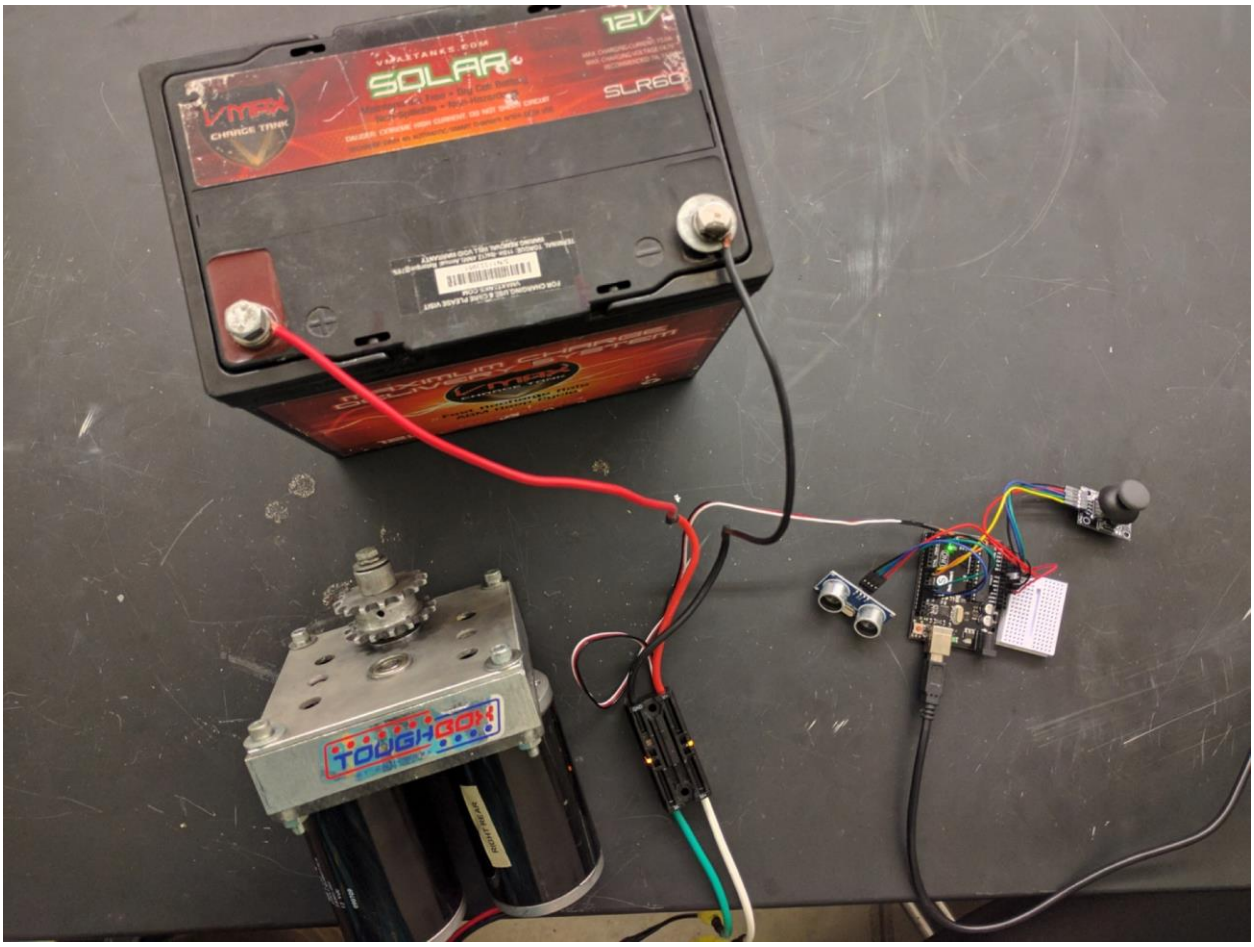

Figure 8: Motor Drive Components

8.1.4 Cameras

Team 14 uses two Raspberry Pi cameras that were donated to us. The cameras have a 5MP resolution which is good enough for our purpose. One camera is used for stop sign recognition and the other for a live video feed from the vehicle and in order to process these 2 Raspberry Pis was used for each one. Figure 9 below shows the camera.

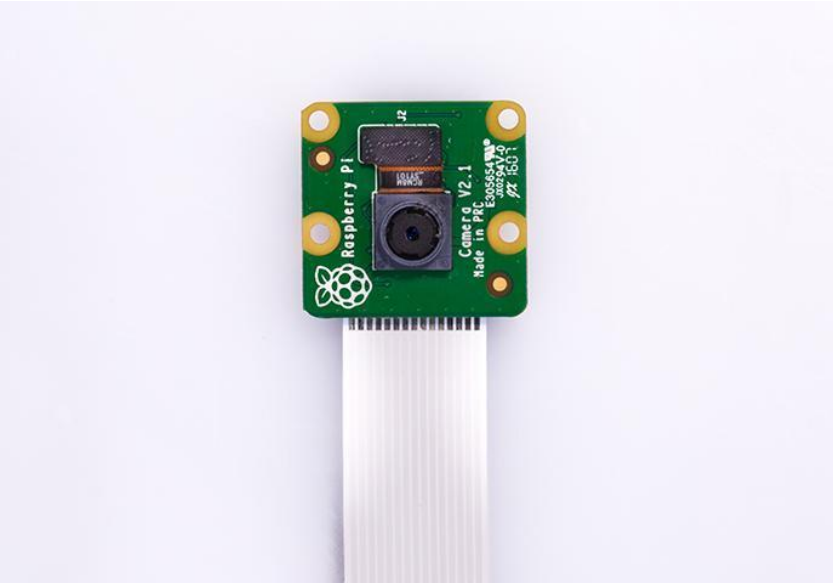

Figure 9: Raspberry Pi Camera

8.1.5 Ultrasonic Sensor

The team added ultrasonic sensors as an inexpensive redundancy safeguard. Ultrasonic sensors use a high-frequency sound and listen for the time for an echo to return in order to determine distance. Ultrasonic sensors are relatively cheap and Peter Ye had experience working with ultrasonic sensors with Arduino. Figure 10 shows the specifications of the ultrasonic sensor used.

Figure 10: Ultrasonic Sensor

8.1.6 Pixhawk controller

The Pixhawk controller used is a flight controller mainly used for drones which can be adapted to control a variety of different vehicles. The controller receives signals from a GPS and compass and sends out pulse width modulation signals to the speed controllers according to the waypoints it generates. It provides a lot of parameters that can be tuned for different vehicle specifications to achieve smooth autonomous driving. It can also connect with a remote control and give the user the option of manual control of the vehicle. Figure 11 below shows the ports of the board.

Figure 11: Pixhawk Control Board Layout

8.1.7 IR speed sensor

IR speed sensor uses infrared to detect anything in front of it. It can also use a potentiometer to adjust how far the sensor can detect. Currently, the speed sensor is set to detect at a range of 3 cm. IR speed sensors are put near each front wheel to measure the speed the vehicle runs at. It measures the time it takes for the wheel to make one revolution and thus give the speed of the vehicle.

Figure 12: Arduino IR Sensor

8.1.8 Raspberry Pi 3 Model B

The team used two raspberry pi 3 model B control boards, one for LIDAR processing and stop-sign recognition and the other for live video feed. The specifications of the board can be found here. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
The reason for having a separate raspberry pi for streaming was that the stream used a different video encoding library that conflicted with the stop-sign recognition. The problem addressed can be found here. https://ubuntuforums.org/showthread.php?t=2353304

8.2 Software

8.2.1 LIDAR and Camera

The LIDAR unit code was procured by Peter Jung who used the ROS Library to pull in raw data and process it into structured data that we can use to avoid obstacles. With the help of the LIDAR donating party, Team 14 has coded with python, C, and bash to interpret the data coming out of the LIDAR and the camera to stop when there is an obstacle or a stop sign on the road. Since the data from these sensors needed to be processed quickly, “NumPy” arrays were used so that the raspberry pi could process data without delay and then stop the vehicle in time to avoid accidents. The referenced libraries are included as [3], [4]. As for the Raspberry Pi, the Raspbian Jessie was used to run the two Raspberry Pi units downloaded from this website[5].

8.2.2 Arduino

The motor driving code was written using the Arduino programming language. This language is a set of C / C++ functions that can be called by the unit. The Arduino can drive the motors at variable speeds. The code changes the throttle and turning output signals from Pixhawk to signals to each motor’s speed controller. The Arduino gets special inputs from the Raspberry Pi when the camera detects stop signs or the LIDAR detects obstacles. Lastly, it also receives input from ultrasonic sensors in the front of the vehicle. The final Arduino code is included the Appendix A.

8.2.3 U-Center

U-center is a free software that comes with the DGPS unit. It can set the protocols of the packages coming in and out of the DGPS units. It also shows how many satellites are connected currently and the health of the signals. Figure 13 shows the interface of u-center.

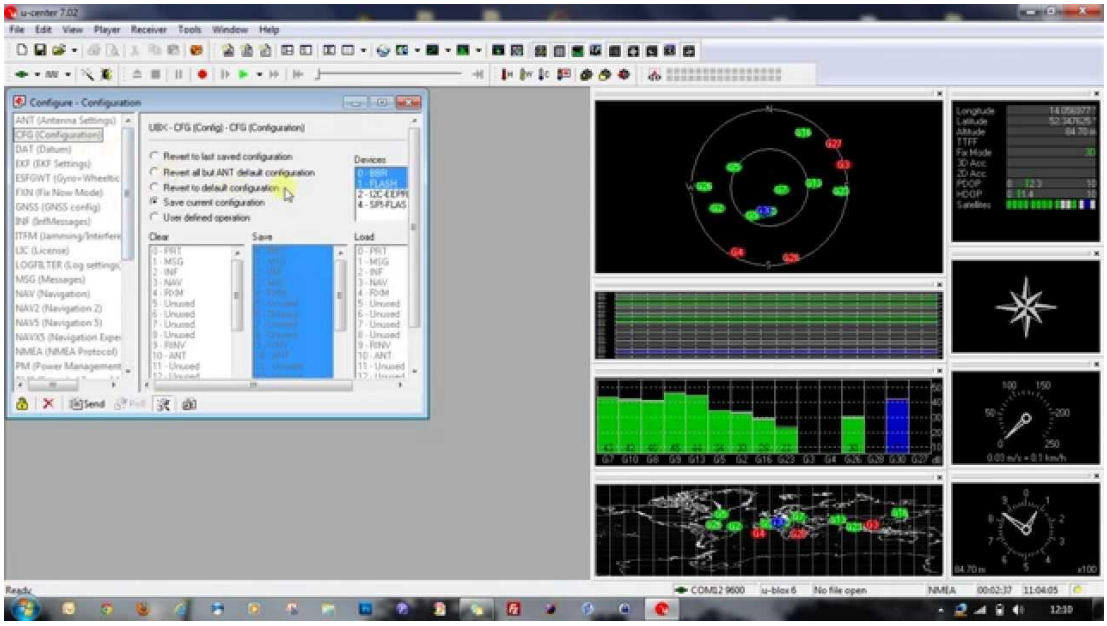

Figure 13: U-Center interface

8.2.4 QGroundControl Software

This open source software is used for Pixhawk board. It has many parameter values users can adjust in order to let a vehicle drive smoothly since vehicles can have different sizes and speeds. The parameter tuning can be found here. It Gives users the capability of putting waypoints on the map to plan a path for the vehicle as shown in Figure 14 below. It has a limit of 700 waypoints which is more than enough for the vehicle to go around the Calvin loop.

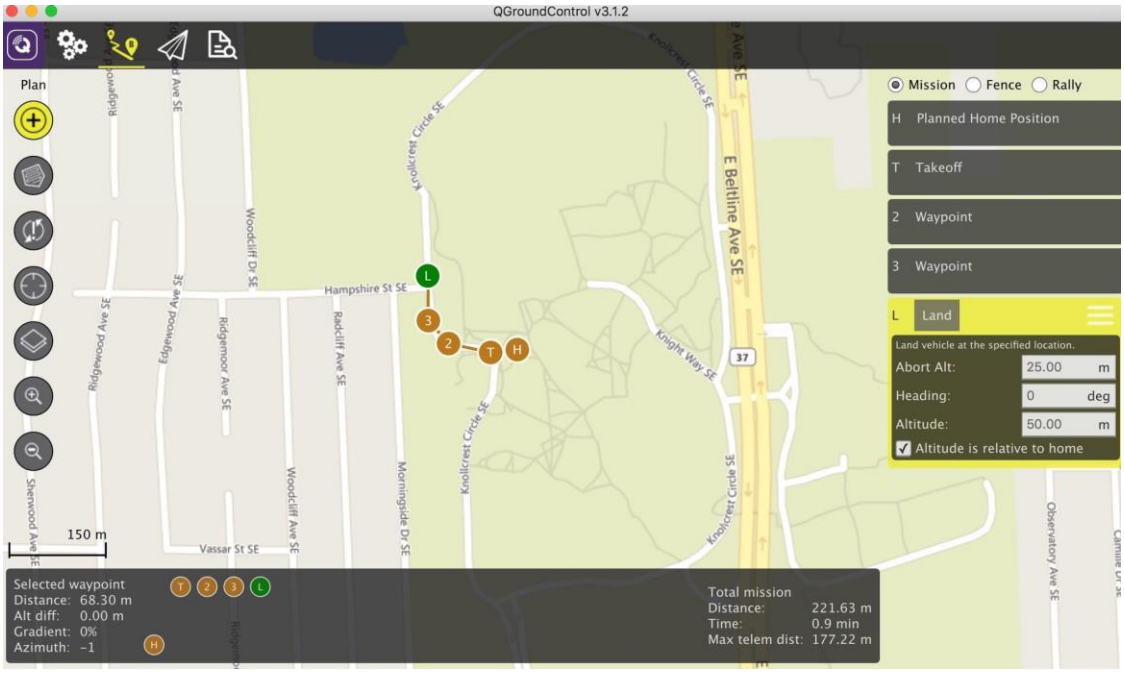

Figure 14: QGroundControl Software Interface

8.3 Motor & Chassis

The motor and chassis are mostly a rework of what the RoboSnow team developed last year. It uses two coupled motors in order to drive the wheels. The wheels are a combination of types. The chassis has been stripped down to the frame so that testing could begin (see Figure 15). The recycled frame was a starting platform for the Auto-bot, however, it was later modified to accommodate design criteria (see section 9.1). The Auto-bot has a gearing system to optimize the top speed of 16 km per hour. Auto-bot shell is designed for optimal functionality of the implemented sensors. The structure also needs to be designed to have enough space for power to completely drive around campus. The purpose of going with a simple design is so testing can begin over interim instead of worrying about making the vehicle a complete and final product.

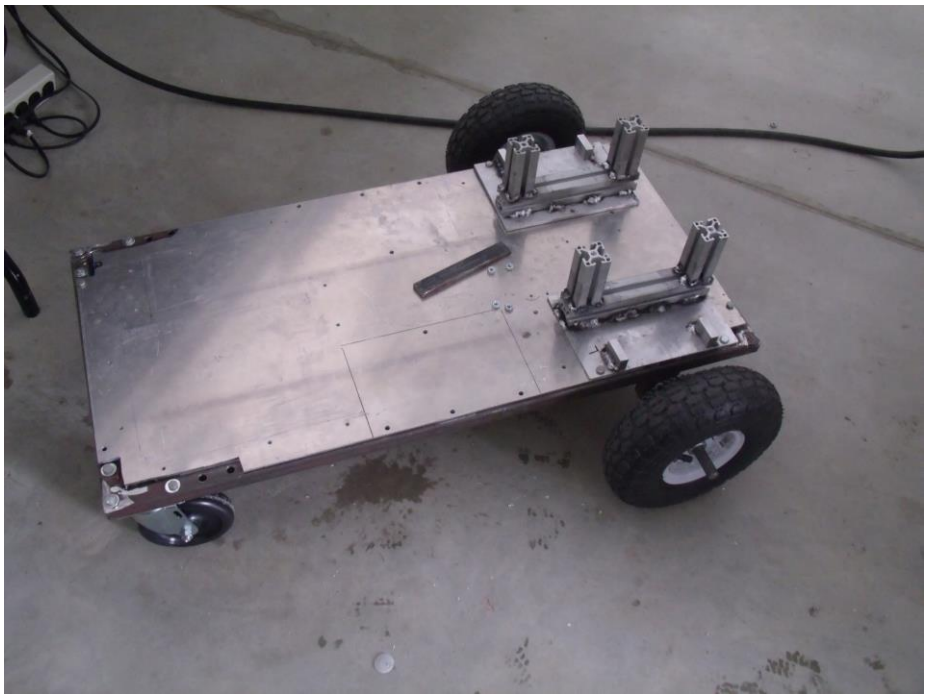

Figure 15: The Original State of the Vehicle

8.4 Interconnections

The final connections of the components are best summarized by the overall architecture diagram below in Figure 16.

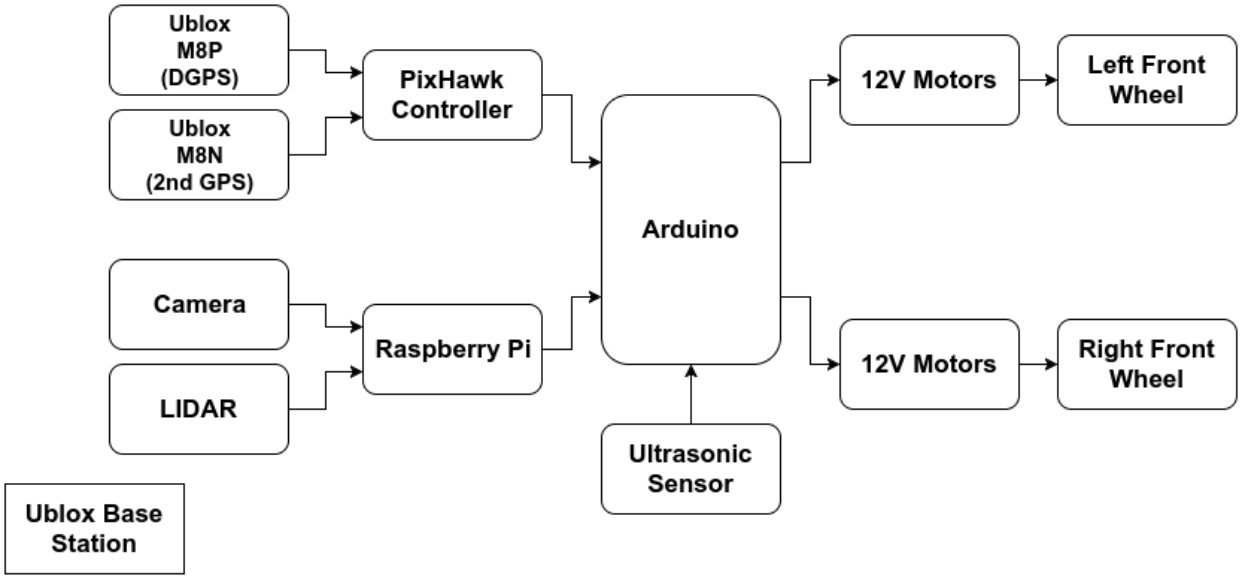

Figure 16: System Architecture

The Rover side of the Ublox GPS unit utilizes a jumper wire connection with the Pixhawk flight controller unit onboard the vehicle. The Ublox base station was meant to send corrections to the rover station of the Ublox which the Team did achieve. However, due to some technical difficulties, the Ublox rover was not allowed to receive corrections and convey information to the Pixhawk flight controller. This is described in detail in section 10.2. Effectively the result was that the base station did not fulfill its duty and the Team had to opt to use optimized settings on the Ublox M8P instead of operating in RTK, or real-time kinematics, mode. There is another DGPS unit that is documented to be compatible with this flight controller and would correct these issue. This unit is described in section 10.2 as well. The Pixhawk controller in Figure 16 also has a secondary GPS (M8N) that serves two purposes. First, it acts as the primary compass since it has a built-in compass. Second, it replaces the M8P in giving coordinates in the event that the M8P fails to get a connection. The Pixhawk has tunable parameters as described in 8.2.4 and uses the software QGroundControl in order to send out a Pulse Width Modulation to the Arduino Uno.

There are a variety of sensors that feed into the Arduino microprocessor. The microprocessor has a direct connection to two ultrasonic sensors that are set to trigger once the vehicle has something within a half meter of the front of the vehicle. The other sensors communicate through the medium of a Raspberry Pi microprocessor. The Raspberry Pi microprocessor takes all the raw data from the camera and the LIDAR unit and sends timed logic signals to the Arduino unit based on the data.

The Arduino Uno is the decision maker in driving the vehicle. This microprocessor uses input from the sensors in order to avoid collisions and input from the Pixhawk in order to determine its motor speeds. The motor speeds are given individually to each motor. Each set of 2 motors on the left and right are given a value that is a ratio, depending on whether it is turning or going straight. The ratio tuning was done through the Arduino code and was optimized to the system's mechanical attributes. For example, the tightness of the chains and responsiveness of the motors was considered and the Team used extensive testing in the manual control mode in order to optimize turning at the desired speed. The desired speed was limited by tuning the values of throttle in the speed controller. The rover is front wheel drive and uses bicycle wheels.

9 Design Criteria, Alternatives, and Decisions

The first major item in the design specifications was to define the needed minimal components in order to get the vehicle functioning and making decisions on its own. Since making an automated vehicle can be extremely complex and very expensive, these two factors were the major restricting factors. Another important factor is the availability of support for the products used since these components are naturally quite expensive and one product failure could result in the system not working and surpassing budget.

In addition to all these technical considerations, it is important to acknowledge the design norms that are major factors in the decisions made. Two of the design norms that stand out in a complex system like this one are transparency and caring in the design project. In the case of designing this car, it is important to work together to solve problems and support one another developing a level of care for each other. When working with companies and outside sources, the same care and cooperation must be extended. The limitations of the system being created and the limitations of the Auto-bot must be clear in the case that future design teams may expand upon the capability of the system. Users also need to understand that this is an attempt to replicate some of the successes of a self-driving car on a very small scale and will not have the comparable safety considerations as automotive companies. Transparency in operation of any automated project is very important because the user needs to understand the decision-making process of the automated vehicle in order to determine how much trust can be placed in that product. Safe testing environments are important for prototyping this product since it is a moving vehicle with no human operating that can stop it immediately.

The nature of automated vehicles is that they typically require an immense amount of capital and time investments which exceed that of a capstone undergraduate engineering project. In response, the Auto-bot project focuses on making a proof of concept version and this is significant because it means that special considerations must be made on factors like price, scale, and quality. Finding system components at low prices is reliant on the ability to define a proper scope by limiting the needs of the implementation. For example, industry camera detection algorithms are very complex and could be a senior design project unto themselves so the team is avoiding using them as a primary data input.

9.1 Wheels and Frame

The mechanical reasoning for the frame and wheels were a consideration especially considering the team decided to recycle last year's snow plow machine. The main difference for the wheels were the parameters the Auto-bot required compared to last year's snow plow. The main factor being the speed of the vehicle. The wheels that came with the snow plow vehicle had vibration issues at the speeds required for the Auto-bot. Bike wheels were the desired solution because they were capable of functioning at the speeds desired and already had a gear integrated into the assembly. However, with changing to the bike wheels the frame needed to be modified in order to have load bearing point on each half of the axle.

Figure 17: Load Distribution

9.2 Sensors

The Auto-bot rover needs capable sensors to meet the requirements of the project. In order to ensure as much safety as possible, this means striving to cover every area around the vehicle and including redundancies in the system to avoid safety issues in the case of part failure. The sensors used in the making of this car are a camera unit with detection software, a LIDAR unit with up to an 8-meter programmable range, and a set of ultrasonic sensors to keep the car from getting too close to other objects. The criteria needed were: to be able to see stop signs and see bigger objects like people and cars.

The camera unit used allows the car to continually scan for a stop sign coming up. The camera, along with the LIDAR, was donated to the team by Paul Vander Kuyl. The camera used is a pi camera.

The camera can detect the stop sign from about 4 meters away, which upon testing gave the vehicle more than enough time to stop. In future years it would be feasible to use similar techniques and work with line detection algorithms in order to provide additional “eyesight” to the rover.

The camera code development was mainly accomplished by Peter Jung using Cascade Machine Learning techniques on a Raspberry Pi 3B. Some limitations of the camera are that it will not detect well if it is too dark outside. The camera detection program was something that the team had saw as the clearest way to make sure that the car saw stop signs. The team was able to find a secondary use of the camera by streaming a video feed to our website so that the rover in action can be seen on camera.

There are not many alternatives that can replace vision detection in this instance since it is hard to recognize a stop sign except through the shape and colors. The team used camera detection because it matched the need while if we had used, for example, LIDAR then we would have no way of telling if the object on the side of the road was a sign or just some other stationary object.

The final camera stream can be found here while it is a live stream (insert link once finalized by Peter). The camera would detect stop signs at a range of 4 meters. It was mounted specifically so that it would cover the side of the vehicle from which it would be able to see stop signs.

9.3 Positioning System

A few very important criteria are directing many of the choices in this project. While keeping things low price to fit within the scope is important, Team 14 must also consider the accuracy of the system. Accurate direction for the vehicle was the top priority in component selection. This is because, with a high precision input of where the vehicle is on a predefined course (like the Calvin loop for example), the user, can tell the vehicle exactly where the road is supposed to be. This means that team Auto-bot can create a low-cost alternative by knowing the environment precisely and telling the vehicle exactly where it should be on the road. This means that the team must program a route beforehand and the vehicle can run it with minimal error.

Table 3: Positional System Decision Matrix

Positioning System Decision Matrix											
Criteria	Price	Weight (4)	Accuracy	Weight (5)	Reliability	Weight (3)	Ease of Use	Weight (2)	Variety	Weight (1)	Sum Total
GPS	10	40	2	10	4	12	8	16	8	8	86
DGPS	**2**	**8**	**10**	**50**	**8**	**24**	**8**	**16**	**5**	**5**	**103**
Local Wifi	4	16	7	35	8	24	6	12	3	3	90
Camera Local Line Detection	5	20	4	20	2	6	1	2	4	4	52

Positioning and understanding where the vehicle is at all times are vital to directing where the car should go. There are many ways to implement positioning systems. One major objective was to keep a car on the right side of the road for this project and keep it operating safely. Top weighting priority was placed on the Price and Accuracy sections since a cost-effective way to determine exactly where the vehicle is at. A medium weight is placed upon the Reliability of the system since Team 14 planning on operating the system in only predictable situations with clear weather and signals. Lower weights are placed upon the variety of products and implementation methods since marketplace options are a nice way to update the system but not extremely important when the design objective is to make the vehicle functional. The weight on Ease of Use refers to whether the system is very complex to implement. This was necessary to include especially for the camera detection method because the complexity of detection algorithms would not have been completed in a timely manner and would hinder progress in the actual integration of the system.

Table 4: Decision Matrix for DGPS

Criteria	Price	Accuracy	Reliability	Ease of Use	Total
weight	4	5	3	2	
Trimble DGPS	4	9	9	2	92
z-survey	1	9	7	5	80
DGPS OLI	7	6	2	1	66
Garmin	3	6	9	2	73
u-blox	7	10	8	9	120

Using Table 4, Team 14 was able to pick out the best DGPS in the market. The team chose the U-blox DGPS because it had the best Accuracy and Ease of use, while it had the best price in the market.

The final Decision to go with the U-blox GPS was made and the system is now integrated with the Pixhawk flight controller so that the rover can successfully pinpoint its location well enough that it will stay on a rather precise course with accuracies up to 2.5 cm. The base station coordinates were

surveyed in at the large rocks north of the North Hall buildings in the middle of the circle. The navigation system setup requires both the rover station and base station fully operational in order to get accurate readings of the coordinates. The weather must also be optimal in order to avoid the GPS losing connection. This still suits the purpose of this project because the Team was not intending for it to be all-weather application scenario in the requirements.

9.4 Controllers

The controllers are extremely important to give commands to the motors so that the rover can drive. The team needed a way to use send a signal to the motors in order to actuate vehicle movement.

The speed controllers used in the decision-making process are shown in Figure 18. The biggest concern was to have something that is properly rated to have at least 54 Amps of continuous output to the motors and was affordable so that a backup could be purchased and be ready to go in case of emergency. The team used Victor SP to control the motors since the Victor SP can support 60 Amps continuously and 100 Amps peak. The ratings were appropriate and the affordability of these speed controllers was aided by the fact that the Team was given two of them from the RoboSnow project. The speed controllers handle the Pulse Width Modulation sent from the Arduino Uno and send it on to the Motors. Team 14 has in its inventory one extra controller because speed controllers can easily burn out.

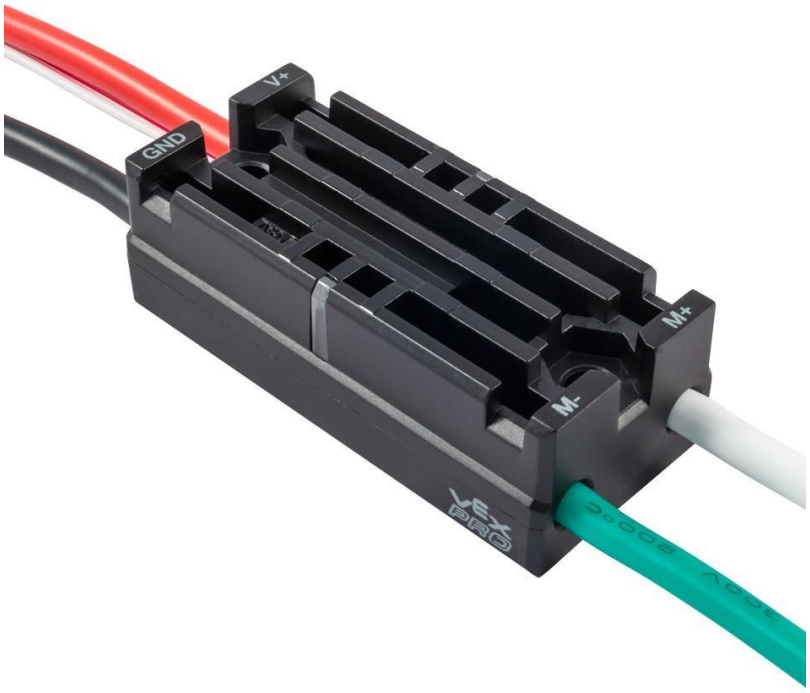

Figure 18: Victor SP Motor Controller

9.5 Motors

The motors used also come from the RoboSnow team. The motors were good enough for the team's project because the application was to create a tank steered rover that would travel never faster than 16 km/hour. The motors can be seen in Figure 19. Each wheel has two motors connected to it to give enough torque so that the vehicle can go 16 km/hour.

Figure 19: First Robotics 12VDC Motor

9.6 Processing Unit

The processor Raspberry Pi 3 Model B unit is used in order to process the data from sensors and an Arduino board is used to drive the motors. The Raspberry Pi was used because it came with the LIDAR unit and the team had familiarity with working with Raspberry Pi computers. Peter Ye worked on the Arduino Uno board and its connections, who had much experience programming with an Arduino board. There were some other microcontroller options but they had no significant advantage over the processors we had. Another option the team had considered was the possibility of using a laptop or phone to increase the processing power available, but once testing began the Team had no major issues that required those alternatives. The decision to use two separate processors to split the workload was a key in success, since the use of the Raspberry Pi 3B for photo scanning and LIDAR used approximately 70% of the CPU during normal operation. Another Raspberry Pi 3B was integrated for a live stream feed.

9.7 Power Considerations

In order to run the vehicle, the team needed to provide power to the components like the motor and the different sensor modules and controllers. So the vehicle would need to provide a steady amount of power and, as a goal, the team wanted the vehicle to at least be able to drive about 10 minutes in our initial project requirements. Failure to provide power to the vehicle and its motors safely could cause a lot of safety concerns. The power provided to the flight controller and sensors must also be consistent. In addition, a safe way to be able to take over control and turn off the rover's power is required.

There are a number of ways to provide power to the rover that the team considered last fall. The type of supply needed was a battery since the rover is going to be driving around outside and should not have any cables dragging behind it. The team weighed the option of pursuing more green technologies like solar power but in the end decided to use 12V Power Cell batteries because it was donated and was the decidedly cheaper route that would not have any significant downtime (for recharging). The team also recycled a power bank from the RoboSnow team from last year. Another power bank with similar ratings was purchased because it perfectly matched the needs of the sensors and processors chosen for this project.

The final battery and power banks are pictured below in Figures 20 and 21. After some testing, it was determined that the limiting factor in operation were the power banks which needed to be recharged after about an hour of use. The purchase of the second unit allowed the team to test for longer periods of time since the units could be swapped out and one could charge while the other unit ran the vehicle.

The reasoning behind splitting the power supply into two separate pieces was two-fold. The first was to avoid demanding too much of the bigger car battery. The second and more significant reason was that it was convenient to have the components that run software left on while the motors were powered off. This was especially useful for the team because the time to make sure all the software was running correctly before testing was rather long before each test. It was extremely useful to switch off the motors so that we could push the cart whenever a problem arose without worrying about damaging the motors and chains.

The team also used a Turnigy 2450 mAh LiPo battery, shown in Figure 22, early to power the speed controllers. Power was feed in the Pixhawk controller and passed on from that controller component to the other microprocessors and sensors. Some components that were not in the vehicle were

powered by a computer through USB ports. Table 5 summarizes what each system unit was powered by.

Table 5: Power Supply Flow

Unit	**Power Provision**
Pixhawk	Power Banks (see Figure 18)
Motors	Car Battery via Distribution Board
Lights	Car Battery via Distribution Board
Arduino Boards	Power Banks
Ultrasonic Sensors (x2)	Arduino Boards
Raspberry Pi Boards (x2)	Power Banks
Lidar	Raspberry Pi Boards
Camera (x2)	Raspberry Pi Boards (x2)
Rover Ublox M8P & M8N	Arduino Boards
Base Station M8P	PC via USB connection
Radio Telemetry	PC via USB connection

Figure 20: 12V Vmax Battery Supply

https://www.vmaxtanks.com/assets/images/slr60.jpg

https://www.amazon.com/AmazonBasics-Portable-Power-Bank-000/dp/B00LRK8JDC

Figure 21: Power Bank for Flight Controller

Figure 22: LiPo Battery for controllers

https://hobbyking.com/en_us/turnigy-2450mah-3s-30c-lipo-pack.html

10 Integration, Testing, and Debugging

The testing of the system began with component testing to determine capabilities of the system. The components were wired together and connected to a processing unit that communicated the software directive to the motors that actuated the vehicle. Integrated testing took place as needed to determine how these systems work together. It was essential that stopping for obstacles had a greater priority than moving toward for example. There are three four categories of tests that the team did, unit test, integration test, functional test, and robustness test.

10.1 Unit Test

10.1.1 LIDAR Test

The LIDAR unit would be tested for its range and the different obstacles it can detect. Since the LIDAR uses a laser to detect things using reflective technology, so unreflective objects might be hard to detect with the LIDAR. So ultrasonic sensors are added to the vehicle so that they detects things that LIDAR might not pick up. The team tested the LIDAR inside and outside of the room. The team tested how fast the LIDAR can pick things in its view which is less than a second. The team also tested the range of the LIDAR and found out that it's less 8m. The team had a problem with the LIDAR reading when testing it outside at night. But it worked fine during the daytime. Right before senior design open house, Peter Jung who was working on LIDAR couldn't connect LIDAR with the raspberry pi. The LIDAR can't be detected by the raspberry pi which may be caused by the connector or inner circuit, so we ended up not being able to have LIDAR working.

10.1.2 Camera Test

The team also tested the camera with stop sign detection. The team first used stop sign pictures on the phone to test how well it responds. It could pick up the image with a delay of about one second. It also depended on the angle of the image and how bright the environment is. While tested outside, the same issue occurred. Vehicle needed to face the stop sign pretty straight in order to recognize it and it can't be farther than 3.66 m away from the stop sign. The environment shouldn't be too bright or dark. This posed constraints on when the vehicle was able to be run. One of the problems the with the Pi camera was that it would not load properly into the Raspberry Pi. To fix this the team added this code: "sudo modprobe bcm2835-v4l2," to "enable" the camera for OpenCV automatically. The code above loads the necessary

drivers to handle everything automatically (e.g. loads the appropriate interfaces (v4l2 drivers) for the raspberry camera). After this code was loaded the Pi camera worked again to detect stop signs.

Near the end of the project, Peter Jung also worked to add another camera for a live video feed from the vehicle that would post to Youtube. The camera used another Raspberry Pi to do the processing. It used school's guest wifi since it can't connect Eduroam (Calvin's internet) because of firewalls. There is a ten-second delay for the video feed because it uses YouTube live feed to show the video, which has a 10 seconds mandatory delay. The team experienced problems with live video feed when the vehicle was running on the roads since it couldn't get wifi signal all the time. The team had to use Josiah's cellphone's hotspot to do the video feed which did not provide enough bandwidth to consistently stream without dropping frames.

10.1.2 DGPS Test

The team put a lot of concentration on the calibration of the DGPS to make sure they can get the most accurate measurements of position. The accurate measurements would make sure that the car is in the right lane. It should also inform the car about the pathing and the destination so it knows its route. The team tested the DGPS boards outside in the circle in front of Spoelhof gym and did survey-in mode on the base station. After it knew its coordinate with an accuracy of .5 m, then the team programmed the two boards to "talk" with each other and reach RTK (real time kinematics) mode on both boards.

The board was tested and showed a high accuracy and responsiveness. The accuracy difference was clear when compared the quality of the survey. Below in Figure 23, the base station setup that Levi Dobson did outside the Engineering Building on the north side lawn. In the U-center software interface, the time and required accuracy of the survey could be set. Once the board fulfills both the minimum time and the set 3D standard deviation are met, the program is able to enter "TIME mode." A successful half meter test screen is shown in Figure 24. The required settings which the team saved in order to operate the boards are linked in the Reference at [6, 7]. These text files can be uploaded to the boards through the U-center interface via the "Tools" section. This and the reasoning behind the settings are explained in the U-center user manual, which can be found in [7]. One important limitation that is explained further in section 10.2 below is that the desired result was only achievable at 9600 Baud rate because the boards could not support RTK operations at any higher rates.

Figure 23: Setup of survey-in base station

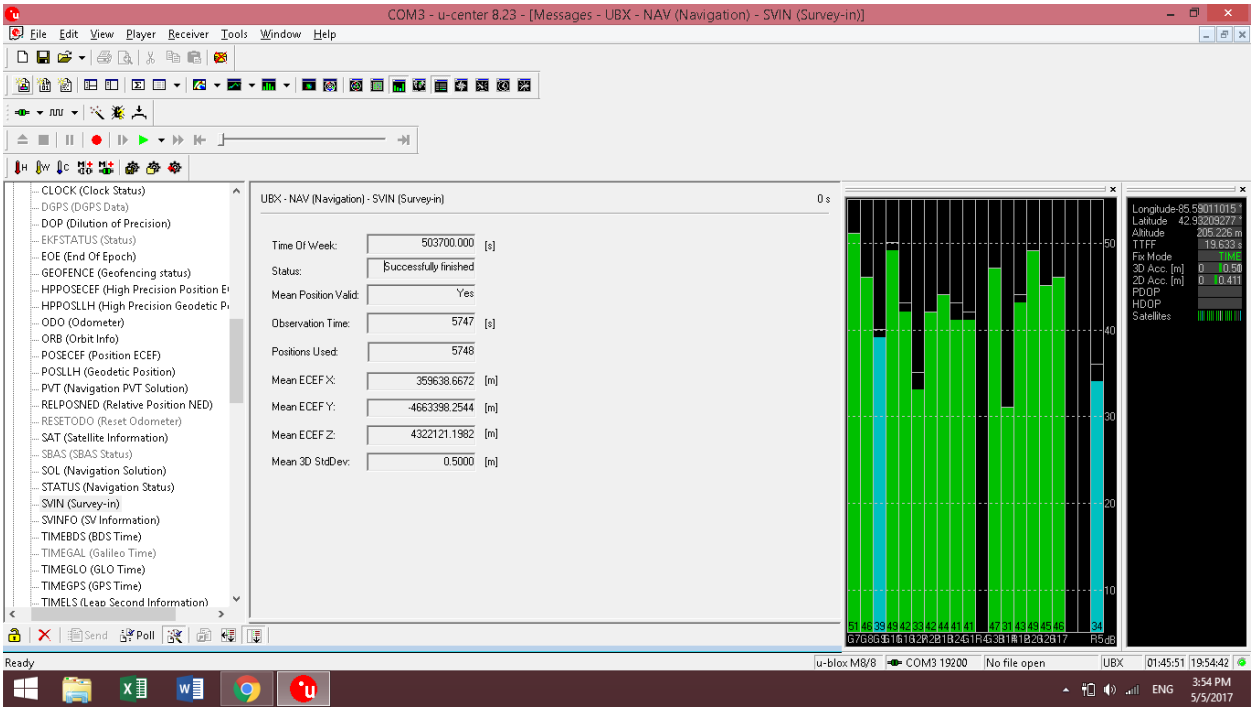

Figure 24: Successful Survey example of .5 meters

10.2 Integration Test

Many tests were taken to make sure all the components worked in synchronization and had no problems talking with each other. This meant that the vehicle could drive in auto mode, starting and stopping at designated places. It also stopped in front of the stop sign and would stop if any obstacle came within a 3m radius of the vehicle.

The team tested the integration of the DGPS with the Pixhawk. This was one of the major obstacles the team anticipated in the beginning of the project because it was critical to the success of the project and the documentation was limited. The DGPS C94-M8P board was released in early 2016, so it was hard to find many resources through research about the connections. First, the team had to change the physical connectors in order to let the Pixhawk and DGPS communicate with each other. The rover DGPS protocol settings conflicted when the team tried to set the DGPS in the RTK mode and communicate with Pixhawk at the same time. The major conflict was that the Pixhawk would automatically reconfigure the rover station board to emit data at an 115200 Baud rate over UART1. This meant that the DGPS would function only as a normal GPS because the board was required to be at 9600 Baud in order to send

corrections. These limitations were later found through research and the team found other people had similar issues, as seen in the Ublox forums [8]. Unfortunately, this meant that the team had to simply optimize the GPS settings for regular operation and test when the weather conditions involved clear skies. The team has located a replacement DGPS that is specialized to support the required Baud rate of 115200 with the Pixhawk while still reaching RTK mode (information can be found in the references section[https://drotek.com/shop/en/drotek-parts/812-3dr-solo-rtk-gps-neo-m8p.html]). This unit uses the same M8P chip from Ublox; but, instead of the C94 application board, a Drotek module board is used in order to achieve that higher Baud rate. Due to the timing of the Team's discovery, a replacement board was not able to be purchased in time and would have to be purchased by other teams in the future should the project be continued by a senior design team at Calvin in the future.

After the Pixhawk can receive signals from the GPS, the next step was to use Pixhawk to control the motors. The team tested the integration of Pixhawk with motors through an Arduino Uno board. The Pixhawk outputs two signals, one as a throttle value and the other as a turn value. The team had to change this information to signals to each motor since the team used using tank steering. In the end, the Team was able to use remote manual control and auto driving to control motors through the Pixhawk and Arduino. The team did problems with the compass coming with the Pixhawk. There is an internal compass in Pixhawk board and an external compass in the M8N U-blox GPS module. The vehicle often had incorrect compass readings during testing. This could have been caused by a magnetic field induced by system wires and motors.

The team also tested the sensor integration with motors. The project used camera, LIDAR, ultrasonic and IR speed sensors. The team had to make sure each sensor working correctly with the motors. The team went outside and tested the camera recognizing stop signs and the vehicle stopped for 3 seconds after the detection. The team also tested that when obstacles came within 2 m of the LIDAR range, the vehicle stopped for 3 seconds. The team had a problem with the IR speed sensors because they often gave incorrect readings of the wheel's RPMs. This can be caused by bad light reflection from the wheels. In the end, the team ended up not using the IR speed sensors.

10.3 Functional Test

After the team put all the components together, the team were able to run the vehicle. At first, the team tested the manual driving function in the room. The vehicle's front wheels were lifted to test safely. Then the team turned on the Pixhawk, Raspberry Pi, Arduino, and lastly the motor's speed controllers. The

team could use the remote controller to increase the speed, turn left or right by pushing the throttle and the roll sticks. Then the team went outside and drove it on the Calvin loop. A problem arose that the front wheels didn't always turn the same speed even though no turn was applied. Another related problem was that when the throttle was at zero which meant braking, the vehicle still moved forward for some distance. Because of the two problems, the team thought of adding speed sensors on both wheels to get motors' actual speeds. Then the team could use the actual speed information to adjust the output signals to achieve the desired speeds. However, as mentioned earlier in the integration test section, the IR speed sensors didn't give correct readings all the time. Peter Ye also figured out that the Victor SP speed controllers had a braking mode which solved the problem of not braking.

The team also tested the auto-driving mode outside of the engineering building. At first, the team tested with three waypoints and for a straight short distance. After changing into the auto-driving mode, sometimes the vehicle didn't start running right away. It was because that the vehicle was far from the starting waypoint and the vehicle was set to start with an acceleration of 1m/s^2. After Peter Ye changed the acceleration to 0 m/s^2, it was able to move automatically. It completed the short straight line mission. Team also tested it around the circle in front of the Spoelhof gym and tested on the Calvin loop for longer distance. The vehicle was able to complete the mission fine, but it wandered left and right sometimes on the road. The two main problem was the GPS and compass. We couldn't get the DGPS to reach RTK mode, so it had an accuracy of about 3m. The compass had a lot of interference from wires and motors, so it was not consistent always.

10.4 Robustness Test

The team tested the robustness of the vehicle base. The team changed the wheels originally from last year's snow plow senior design vehicle base because their design couldn't meet our speed goal of maximum 16 km/hour. Josiah checked at what speed vibrations were occurring in the old team's wheels. The wheels did not meet the specification of 16 km/hour. This made the team opt for children's bicycle wheels because they were not likely to vibrate at higher RPMs like the previous wheels used. The casters were also tested for vibrations with agile direction changes. The solution was to move towards soft rubber caster and to place the weight of the battery overhead the casters enabling the vibrations to be pushed out of the speed range the vehicle travels. In the end, there was still some vibration in the caster wheels, but it only occurred at higher speeds.

11 Conclusion

The major component of the system that made the project feasible was the differential GPS unit. After looking at options within the project budget, it was determined that the NEO-M8P-2 from U-blox would be the high precision GNSS, or Global Navigation Satellite System, selected in order to drive the system. Additional units to direct car motion will be a LIDAR unit to detect distances from obstacles, a camera that notices key pieces of the environment in order to provide a critical stopping warning, ultrasonic sensors to provide redundancy in obstacle detection, and IR speed sensors for providing a feedback loop for setting motor speeds accurately.

Due to the complex and expensive nature of making autonomous vehicles, simplifying the autonomous vehicle to drive in limited environments was necessary. For example, the desire of the project is for the vehicle to avoid accidents while maintaining a course. In order to have the appropriate goals, it was necessary to consider that the scope of the project is to drive it in slow environments, where stopping and waiting for an obstacle to be cleared (which is not feasible on high-speed roads for autonomous vehicles) can be an appropriate response. The driving condition will be in clear skies during the day in order to get good GPS signals and for better stop sign recognitions.

The project is inspired by the latest major trend in the automotive industry of autonomy and has several design benefits and goals. It will primarily serve as a proof of concept project in order to introduce the team to a topic on which a career can be built. The project also is meant to be a demonstration of the advancement of technology to the point where even a small project with a small budget can make a car assign to itself direction and caution.

11.1 Further Development

This project by its nature can have many components added to it in order to better help it make accurate decisions using GPS and upgrading detection coding and sensors is a great goal for the future of this project in order to improve the safety and usefulness of the vehicle.

12 Acknowledgements

Professor Michmerhuizen……. Main Project Advisor

Mr. Glen DeVos……Project Funding Patron

Professor Tubergen……Vehicle Mechanics Consultant

Mr. Bill Haverkamp……...Project Funding Consultant

Mrs. Michelle Krul……Senior Design Administrative Coordinator

Mr. Bob DeKraker……Parts Ordering Consultant

Mr. Eric Walstra……Industrial Advisor

Mr. Paul Vander Kuyl…….....LIDAR donor

Our Parents……...For feeding and raising us

13 References

1. LIDAR information file
   https://www.robotshop.com/media/files/pdf/datasheet-rplidar.pdf.
2. PWM driver board library
   https://learn.adafruit.com/16-channel-pwm-servo-driver/library-reference
3. Cascade Classifier Machine Learning.
   http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html
4. ROS rplidar package that was used for LIDAR.
   http://wiki.ros.org/rplidar
5. Raspbian Jessie
   https://www.raspberrypi.org/downloads/raspbian/
6. Config File from PixHawk

   https://github.com/ArduPilot/ardupilot/tree/master/libraries/AP_GPS/config

7. Config File Saved from Levi's PC

   Rover:

   https://drive.google.com/open?id=0B2toltIbFLgtaEhxVk1vMGlqNGc

   Base:
   https://drive.google.com/open?id=0B2toltIbFLgtc25JNmpRVE1JNEE
   User Manual:
   https://drive.google.com/open?id=0B2toltIbFLgtQTdaX1liTnJuWVk

8. https://forum.u-blox.com/index.php?qa=4407&qa_1=c94-m8p-rover-doesnt-receive-rtcm-corrections

14 Appendices

Appendix A

Arduino Code

```
//#include <TimerOne.h>
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

// called this way, it uses the default address 0x40
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

#define trigPinr  13  //ultrasonic sensor trig pin
#define echoPinr 12   //ultrasonic sensor echo pin
#define trigPinl  11  //ultrasonic sensor trig pin
#define echoPinl 10   //ultrasonic sensor echo pin
const int Joystick_INPINX = 0; //joystick pin to A0
const int Potentialmeter_PinIn = 2; //Potentialmeter pin to A2
const int CameraPIN = 9; //camera digitalread pin
byte PWM_PIN1 = 8; //pixhawk pin 3 for throttle
byte PWM_PIN2 = 7; //pixhawk pin 1 for turn

const int IRPin2 = 2; //IR sensor INPUT
const int IRPin1 = 3; //IR sensor INPUT

int pwm_value1 = 0;
int pwm_value2 = 0;
int pwm_out1 = 0;
int pwm_out2 = 0;
int x_position;
int potentialmeter;

int pwm1;
int pwm2;
int vehiclespeed = 0;
int cameravalue = 0;
int LIDARvalue = 0;
int pwmMIN = 2270;

long durationr, distancer;
long durationl, distancel;
float counter1=0.0;
float counter2=0.0;
```

```
uint8_t servonum1 = 0; //left motor
uint8_t servonum2 = 1;//right motor, value needs to higher than left by 10


void setup() {
//  pinMode(IRPin2,INPUT);
//  pinMode (IRPin1, INPUT) ;
  pinMode(Joystick_INPINX, INPUT);
  pinMode(Potentialmeter_PinIn, INPUT);
  pinMode(PWM_PIN1, INPUT);
  pinMode(PWM_PIN2, INPUT);
  pinMode(trigPinr, OUTPUT);
  pinMode(echoPinr, INPUT);
  pinMode(trigPinl, OUTPUT);
  pinMode(echoPinl, INPUT);
  pinMode(CameraPIN, INPUT);

//  Timer1.initialize(2000000); // set timer for 1sec
//  attachInterrupt(0, docount1, FALLING);  // increase counter when speed sensor pin goes low
//  attachInterrupt(1, docount2, FALLING);  // increase counter when speed sensor pin goes low
//  Timer1.attachInterrupt( timerIsr ); // enable the timer

  Serial.begin(9600);
  pwm.begin();
  pwm.setPWMFreq(400);  // Analog servos run at ~0 Hz updates
//  yield();
}

//void docount1()  // counts from the speed sensor
//{
//  counter1++;  // increase +1 the counter value
//}
//
//void docount2()  // counts from the speed sensor
//{
//  counter2++;  // increase +1 the counter value
//}

//void timerIsr()
//{
//  Timer1.detachInterrupt();  //stop the timer
////  Serial.print("Motor1 Speed: ");
//  float rotation1 = (counter1 / 2);  // divide by number of holes in Disc
////  Serial.println(rotation1,2);
////  Serial.println(" Rotation per seconds");
//  counter1=0;  //  reset counter to zero
////  Serial.print("Motor2 Speed: ");
```

```
//  float rotation2 = (counter2 / 2);  // divide by number of holes in Disc
////  Serial.println(rotation2,2);
////  Serial.println(" Rotation per seconds");
//  counter2=0;  //  reset counter to zero
//  Timer1.attachInterrupt( timerIsr );  //enable the timer
//}

void checkUltrasonics()
{
//  if (distancer >= 50 && distancer < 130 ) {
//   pwm_out1 = pwm_out1 - (vehiclespeed/2);
//  }
//  if (distancel >= 50 && distancel < 130 ) {
//   pwm_out2 = pwm_out2 - (vehiclespeed/2);
//  }

  if ((distancer >= 5 && distancer < 50) || (distancel >= 5 && distancel < 50)) {
   pwm_out1 = pwmMIN;
   pwm_out2 = pwmMIN;
  }

}

void loop() {
  pwm1 = pulseIn(PWM_PIN1, HIGH, 25000);
  pwm2 = pulseIn(PWM_PIN2, HIGH, 25000);


  digitalWrite(trigPinr, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinr, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinr, LOW);
  durationr = pulseIn(echoPinr, HIGH);
  distancer = (durationr/2) / 29.1;  //get distance in cm


  digitalWrite(trigPinl, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPinl, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPinl, LOW);
  durationl = pulseIn(echoPinl, HIGH);
  distancel = (durationl/2) / 29.1;  //get distance in cm

//  x_position = analogRead(Joystick_INPINX); //between 0 to 1023
//  potentialmeter =  analogRead(Potentialmeter_PinIn);
```

```
//  pwm_value1 = map(potentialmeter, 0,1023, 2250, 3140); //stop at 2240-2340
  pwm_value1 = map(pwm1, 1050,1950, pwmMIN, 3140);
  pwm_value2= map(pwm2, 1050,1950, -50, 50);

  pwm_value1 = constrain(pwm_value1, pwmMIN, 3140);
  pwm_value2 = constrain(pwm_value2, -50, 50);

  vehiclespeed = pwm_value1 - pwmMIN;

  if(pwm_value2 > 5){
    pwm_out2 = vehiclespeed*(50-pwm_value2)/50 + pwmMIN;
    pwm_out1 = pwm_value1;
    }
  else if(pwm_value2 < -5){
    pwm_out1 = vehiclespeed*(50+pwm_value2)/50 + pwmMIN;
    pwm_out2 = pwm_value1;
    }
  else {
    pwm_out1 = pwm_value1;
    pwm_out2 = pwm_value1;
    }

  //control the PWM driver

  checkUltrasonics();

  if (digitalRead(CameraPIN) == HIGH) {
      pwm.setPWM(servonum1, 0, pwmMIN);
      pwm.setPWM(servonum2, 0, pwmMIN);
      delay(3000);
  }
  else {
      pwm.setPWM(servonum1, 0, pwm_out1);
      pwm.setPWM(servonum2, 0, pwm_out2);
  }

//  Serial.print("pwm_value1  ");
//  Serial.println(pwm_out1);
//  Serial.print("pwm_value2  ");
//  Serial.println(pwm_out2);
//  Serial.print("pixhawk throttle  ");
//  Serial.println(pwm1);
//  Serial.print("vehicle speed  ");
//  Serial.println(vehiclespeed);
//  Serial.print("turn  ");
//  Serial.println(pwm_value2);
```

```
//  Serial.print("potentialmeter  ");
//  Serial.println(pwm_out1);
//  Serial.print("right distance  ");
//  Serial.println(distancer);
//  Serial.print("left distance  ");
//  Serial.println(distancel);
//
//  delay(500);

}
```

Stop sign detection code

```
import numpy
import cv2

def detect(img):
cascade = cv2.CascadeClassifier("some xml file")
rects = cascade.detectMultiScale(img, 1.3, 4, cv2.CASCADE_SCALE_IMAGE, (20,20))

if len(rects) == 0:
        return [], img
rects[:, 2:] += rects[:, :2]
return rects, img

def box(rects, img):
for x1, y1, x2, y2 in rects:
        cv2.rectangle(img, (x1, y1), (x2, y2), (127, 255, 0), 2)

cap = cv2.VideoCapture(0)
cap.set(3,400)
cap.set(4,300)

while(True):
ret, img = cap.read()
rects, img = detect(img)
box(rects, img)
cv2.imshow("frame", img)
if(cv2.waitKey(1) & 0xFF == ord('q')):
    break
```

Appendix B

Task Flow Chart

	Task Mode	Task Name	Start	Finish	Predecessor
1		Reseach- Find Sensors	Wed 9/14/16	Wed 9/28/16	
2		Get a LIDAR unit	Wed 9/28/16	Wed 11/16/16	1
3		Get LIDAR unit reading Data	Wed 11/16/16	Thu 12/8/16	2
4		LIDAR translator data to detect threats	Thu 12/8/16	Sat 1/7/17	3
5		LIDAR integrated into speed controller	Wed 2/1/17	Wed 3/1/17	4,10
6		Reseach- Find Location System	Tue 11/15/16	Tue 11/22/16	
7		Choose a DGPS	Tue 11/22/16	Fri 11/25/16	6
8		Buy a DGPS	Fri 11/25/16	Fri 12/2/16	7
9		Get GPS working by itself	Fri 12/2/16	Tue 1/3/17	8
10		Geo-fencing on GPS	Wed 1/4/17	Wed 2/1/17	9
11		GPS data integrated into speed controller	Wed 2/1/17	Sat 3/4/17	10
12		First Full System Testing	Sat 3/4/17	Mon 3/13/17	11,5
13		Refine System	Mon 3/13/17	Wed 3/29/17	12
14		PPFS Outline	Wed 9/14/16	Wed 10/5/16	
15		Draft PPFS	Wed 10/5/16	Mon 11/14/16	14
16		Final PPFS	Mon 11/14/16	Mon 12/12/16	15
17		Final Design Report	Wed 3/29/17	Sun 5/7/17	16,13,19
18		Introduction Oral Presentation	Wed 9/21/16	Fri 10/21/16	
19		PPFS Summary Presentation	Fri 10/21/16	Mon 12/5/16	18
20		Senior Design Night	Sat 5/6/17	Sat 5/6/17	19

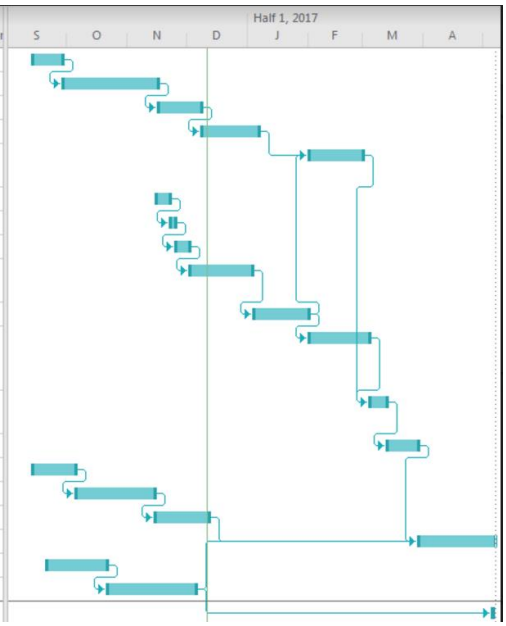

Figure 3: Task Flow Chart