

ECE 480 Final Report

AgBOT Competition

Michigan State University

ECE 480 Senior Design

Team 8

08 Dec 2015

**Sponsor: Michigan State University, Department of
Electrical and Computer Engineering**

Facilitator: Prof. John Albrecht

Manager: Harsh Desai

Web Master: Chen Hao

Presentation Prep: Yirui Wu

Lab Coordinator: Sumanto Pal

Document Prep: Yanqi Wang

Acknowledgement

Dr. Timothy Grotjohn: A special thanks to Dr. Timothy Grotjohn from Electrical and Computer Engineering department, for being a sponsor and allowing us to explore in the field of robotics through the AgBOT challenge. We truly appreciate you for letting us choose the specifics of the project and thank you for your time and support.

Prof. John Albrecht: A heartfelt thank you to Prof. John Albrecht from the ECE department and our facilitator for this project. Your guidance and knowledge has been very helpful for us in completing this project. We are greatly thankful for the time that you took out for us and this project. We truly appreciate your help in completing this project.

Executive Summary

Michigan State University's department of Electrical Engineering is participating in AgBOT competition which will occur in May of 2016. The primary objective of the competition is to develop an Agricultural Robot (AgBOT) that assists farmers in performing field works efficiently.

Team 8 was given the task to do the research over the competition and create a prototype that will help future team (Spring 2016 Senior Design team) in making a Robot that will represent Michigan State University in 2016 AgBOT competition. Thus, team 8, has developed a prototype AgBOT that will serve as the platform for the future teams. The prototype is capable of performing autonomously based on the GPS location and can established a steady communication with the base User Interface.

Table of Content

1. Introduction and Background

- 1.1 Introduction of problem
- 1.2 Background information
- 1.3 Problem Statement
- 1.4 Challenges
- 1.5 Summary

2. Exploring the solution space and selecting a specific approach

- 2.1 House of Quality technique
- 2.2 Exploring the solution
- 2.3 Feasibility and decision matrices
- 2.4 Solution Summary
- 2.5 FAST Diagram
- 2.6 Initial Project Management Plan
- 2.7 Initial Estimate of Budgets
- 2.8 Initial and Final Gantt Charts

3. Technical description of work performed

- 3.1 Robot Skeleton and Motor Transmission
 - 3.1.1 RC car
 - 3.1.2 Motor Driver
- 3.2 Sensor Integration
 - 3.2.1 Hardware (Wiring Method)
 - 3.2.1.1 Navigation sensors (IMU)
 - 3.2.1.2 Accelerometer and Gyroscope
 - 3.2.1.3 Magnetometer
 - 3.2.2 Software (Library Data Association)
- 3.3 Wireless Communication Setup (LAN)
- 3.4 Displaying live video feed
- 3.5 Power distribution module
- 3.6 Protecting Case design

4. Test data with proof of functional design

- 4.1 Design Specifications
- 4.2 Hardware design efforts
- 4.3 Hardware implementation and photo documentation
- 4.4 Software and interface design requirements
- 4.5 Software implementation

4.5.1 GPS-Based Autonomous Navigation Algorithm

4.5.2 Base Station User Interface (Human-machine Interfaces)

4.6 Problem encountered and solution

5. Integrated System Tests

5.1 Test Run 1: Test on the live video feed and the remote control

5.2 Test Run 2: Test on the sensor data acquisition for both GPS and IMU

5.3 Test Run 3: Test on the integrated prototype on a breadboard

5.4 Test Run 4: Test on the final design on a PCB with sophisticated power module

6. Design Issues

6.1 Product Safety

6.2 Lifecycle Management

6.3 Limitation of Design

6.4 Future Improvement

6.5 Power

6.6 Control Loop Algorithm

7. Final cost, summary, schedule and conclusions

Appendix

- 1. Technical roles, responsibilities, and work accomplished**
- 2. Literature and website references**
- 3. Detailed technical attachments**

1. INTRODUCTION AND BACKGROUND

1.1 Introduction

The 21st century is the century of technological advancement and with over 2.11 million farmers, with an average of 434 acre of farming land, only in USA; Farmers should be provided with the best equipment to produce the total yield of around 10 billion ton of crops. AgBOT competition is the fusing point between agriculture and technology. AgBOT is the abbreviation for Agricultural Robot that is designed to perform agricultural field task with minimal human intervention. With the development of the AgBOT, not only will it make the agricultural tasks easy but will do it with the speed. In terms of work force, as you can see in fig 1, in USA there is a decaying trend in employment rate versus increase in the demand in terms of GDP. Agricultural robot will help in filling some of those gaps that is hurting farming process due to under employment. AgBOT will then make farming faster and will eventually help in increasing the quality of the crops along with solving the problem of employment in agriculture area.

1.2 Background

The goal of the project is to build an agricultural robot for the AgBOT competition. The competition's main sponsor is the AirBridge Company. AirBridge designs and provides ultra-high speed internet communication infrastructure for mobile and static environments. The task at hand was to create an unmanned robot that can plant seeds and fertilizer on the field. The task of the competition is to plant an assigned set of GPS coordinates in two 30-inch wide rows. There would be a total of 12 rows, each of which would be half a mile long. The AgBOT should also be able to operate within speeds ranging between 3.5-10 miles per hour. Moreover, the AgBOT should be capable of docking and loading two separate varieties of seed while weighing the intake of seed and sending that data back to the user. On the field, the AgBOT is expected to capture and send real time video back to the user base station. The video needs to be front and back which would give the user a rough idea if or not the robot is performing as expected or having some difficulties. Therefore, there is a need for the user to be able to intervene whenever they desire. Thus, the robot should also have the ability to perform the orders received from the base station. According to the specifications, the user of the AgBOT should be able to control/intervene with the following:

- seed rate
- fertilizer rate
- the speed of the AgBOT
- down pressure of press wheel
- data migration from AgBOT to network host/operator
- Use BATS mobile antenna for communication

One of the requirements of the competition was the use of the BATS wireless system. BATS is a fully developed multipurpose transmission system. It helps in the transmission of data between the robot and the base station for long distances. It supports major parts of the capabilities this challenge is asking for. This antenna is built with military technology and can track up the object up to 50 miles. It also provides high capacity data transmission rate, which is perfect for our GPS navigation and video streaming purpose.

To meet the requirements, sensors needed to be implemented into the AgBOT. Sensors should also be implemented on the robot for various purposes. Some of the sensors required would be for the seed rate (how much seed is left/lost), fertilizer rate (how much fertilizer is left/lost) and also for the down pressure of the wheels (how much pressure is exerted on the wheel when the robot is completely loaded). The robot should also be able to make real time decisions by itself; for example, it should be able to realign itself if it goes off the path or does not move in a straight line. Thus, a vision based control was required for the AgBOT to be able to achieve the desired outcome.

1.3 Problem Statement

The AgBOT competition is a challenge solving event where different group of people with the background in different areas of engineering and agriculture come together and try to solve problems related to agriculture. The theme for the 2016 AgBOT competition is to make an autonomous vehicle that can do the planting process with minimal human intervention.

The team had four months and \$500 budget to work with on the project. Given the size of the competition and time, it would be impossible to make a full scale autonomous vehicle. Although the competition requires that the competing teams implement the BATS wireless system into their model, it was not a feasible option for this project. The main reason being the size of the BATS, which needs to be mounted on the full scale large vehicle that has

capability of carrying substantial amount of equipment. Also, the BATS was quite expensive (~\$16000), and thus we could not buy one. Therefore, long range wireless communication was not possible for this project as there is no other system that can help with long range wireless data transfer.

1.4 Challenge

Under the provided time and budget, it is impossible to produce the full scale autonomous vehicle. Therefore, the team came an agreement to build a scaled down version of the real AgBOT, which the sponsor concurred to. The proposed prototype only focuses on the electronics part of the competition built on a scaled down version. There will be three main areas that the team will be concentrating upon and those areas are:

- Communications
- Independent power supply
- Navigation
- Data processing

Thus, the final model will be a scaled down version of the final AgBOT that will compete in the competition; performing most, but not all of the functions that the competition requires. But, It will simulate the most electrical and computer engineering functions as mentioned above. Moreover, it will serve as a stepping stone for the future teams to build on and complete the AgBOT competition.

1.5 Summary

In the project, the team decided to build a scaled down robot which could simulate the behavior of the AgBOT on electronic performances. First of all, the robot to be built should run on the floor properly with wheels. Secondly, the robot has to be navigated by location commands given remotely. It should calculate the route autonomously and go through different rows on the field. A base station needs to be setup to give commands to the robot. Last but not least, data should be transmitted to the base station from the robot. The robot is supposed to collect live video, speed and the location of itself and send the data back to the base station. Moreover, the robot is also expected to travel in straight rows that are 100m in length. Also, there has to be some kind of remote control, in case the user needs to make some changes in the direction of the robot or otherwise.

2. Exploring the solution space and selecting a specific approach

2.1 Critical Customer Requirement

The scaled down version of the AgBOT needs to achieve the following functionalities:

- Autonomous Navigation
- Communication system
- Passive sensors reading
- Front and Rear view live feed

The Autonomous system should be capable of navigating the robot into six row, each 100 meters long and should be susceptible to any change in the course due to any natural events. Besides the navigation system, the robot should have the capability to communicate with the base station through a steady communication system and should be able to transfer any data gathered by the sensors, such as location, speed and/or heading direction.

	GPS navigation	power supply	video feedback	data collection	movement speed
troubleshoot	8	9	5	5	5
reusable	N/A	8	N/A	N/A	N/A
life time	7	8	8	N/A	N/A
low power	9	N/A	8	6	6
range request	9	N/A	7	7	N/A
Accuracy	8	8	N/A	6	8
cost efficiency	7	9	6	6	6

Table 1: House of Quality Technique

2.2 Exploring the solution

The goal of the project is to build a scale down agriculture robot. It has to be able to go to desired destination based on GPS coordinates and it should also be able to transmit real time video and data back to the base station which is more than 100 meters away from it. The first criteria that the team decided to tackle was the communication system. Since the AgBOT will be operating in farm and over a vast area, the team decided to look at the options like Bluetooth and Wi-Fi based communication systems.

Communication is an important part of this project. In order to navigate and to carry out any data transfer, the communication between the robot and the based station is very important. Thus the main topic of discussion for the team was to find a reliable communication systems that is capable of transmitting and receiving large amount of data and controls.

The group has three proposed design solutions, which are primarily determined by the means of communication. Since the data and video streaming between the car module and the base station has to be done wirelessly and over a substantial range, we had to explore options other than Wi-Fi itself; although Wi-Fi was one of the solutions that the team looked at. Some of the explored solutions were:

	ZigBee	WAN	LAN
Application	Monitoring and Control	Email, Web, Video	Local Network
Data Rate	1 Mbits/s	150 Mbits/sec	1000 Mbits/s
Range	10-100 meters	Worldwide	100-200 meters
Power Consumption	Low	High	High

Table 2: Specifications of ZigBee, WAN and LAN

All the three options that are mentioned in table 1 is capable of providing the steady communication for 100 meters. Thus selection matrix was used to determine which type of system will be used in the design.

2.3 Feasibility and Selecting an Approach

The project was based on a fixed budget. The budget provided to us was \$500. Thus, the use of BATS wireless system was out of our scope as the system in itself would cost us more than \$500. Therefore, after looking at the different options that we had in our hands, we came up with the selection matrix for the communication criteria for our project. It is shown in Table 2 below.

The way the selection matrix works is we assign different points to the different communication options with respect to how well they fulfill the design specifications needed. For example, WAN scored high on the “Signal Reach” category but did not score well on the

“Power Consumption” category. We then add up all the points for each communication solution. The design with the most points will be the best choice for the design.

The three main communication options that were best suited for our project are shown in the matrix, they are:

- 1) ZigBee
- 2) LAN
- 3) WAN

Communication Criteria		Importance	Possible Solutions		
			ZigBee	WAN	LAN
Data Capacity	Streaming Video Quality	4	3	7	8
	Network Latency	4	9	6	8
Range	How far the signal can reach	3	4	9	6
	Trans. Distance between PC and Car	5	4	5	9
Protocol	Communication Stability	5	9	6	7
	Networking Security	3	9	7	8
Compatibility	working with current system	5	1	9	9
	User Friendly	3	3	9	8
Cost	No Additional Parts Needed	2	3	7	7
	Low Power Consumption	3	9	5	5
Totals			199	256	284

Table 3 - Selection Matrix for communication option

It is clear from the selection matrix that the best choice for communication in our project would be the use of LAN.

2.4 Solution Summary

Local Area Network is the preferable choice for the project and the only attribute that separates LAN from Wide Area Network (WAN) is LAN does not need any access to any internet services and multiple devices can be connected wirelessly. Also the LAN can handle the data traffic up to 1000 Mbit/sec and will be sufficient to transmit data along with the video stream.

2.5 FAST Diagram

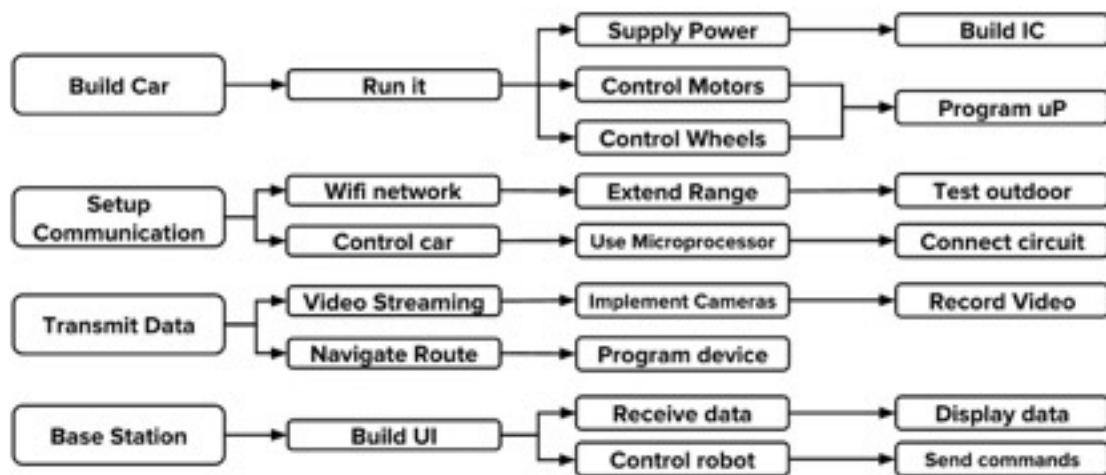


Figure 1: Fast Diagram

2.6 Initial Project Management Plan

The initial project management plan follows the logic of the fast diagram, shown in figure 1, from the left to the right.

Phase 1) the initial step was to meet with our sponsors, and understand the primary objective for the project. Once, understood the project and doing the background research, facilitator was consulted for any advice.

Phase 2) the initial design was then made and primary tasks were been distributed among the team members. Building a car, constructing a wireless network and setting up a base station for user are the three basics to start with simultaneously and separately. These three main parts are going to be tested first at a high priority to ensure the project is able to continue its critical paths. For the car, it has to run some straight functions such as going forward and backward with the power supply in the lab. For the wireless network, it should allow the microprocessor which is the raspberry pi to receive signals from some other devices in the same network. As well, for the base station, it should have an initial user interface for user to input commands to the car.

Phase 3) after the three major steps are finished and tested, they should be combined at a primary level. The first prototype is expected to be a car that can move properly under the control remotely. Up to this moment, the car is supposed to be a robot which can only be controlled by the user and run indoor. The next step of the project is to focus on the independent power supply, autonomous system and data transmission.

Phase 4) the responsible blocks for these are a power distribution module, an integrated sensor system and software codes. The team will work on these parts hard in the second half of the semester to achieve the main functions of the robot claimed in the problem statement & solution section. From the second one to the final prototype, the team is going to implement different function blocks on the first prototype. The last step, in the schedule, is to test the prototype and improve the performance of it.

A Gantt chart, representing the timeframe for a specific task and member associated with it is shown in APPENDIX 3.1. The initial Gantt chart was created according to the primary objective and thus represents the obvious task. As the final design approaches, the Gantt chart became more descriptive with the tasks and also with the timeframe.

2.7 Initial Estimate of Budgets

The initial budget handed to the team was \$500. As the AgBOT was the scaled down version of the actual Robot, the budget provided looked sufficient enough complete the project. The different expenses that were made are documented below in Table 4:

After the initial orders were made, the team was left with \$110 to experiment with any changes that may benefit the final product in terms of performance.

Quantity	Name of the item	Cost/Unit (\$)
2	Raspberry Pi Model 2 Kit (with Cameras)	\$75
1	Adafruit Ultimate GPS Breakout	\$40

1	Maisto R/C Rock Crawler	\$40
1	MPU9150 Gyroscope and Accelerometer	\$35
1	Dual H-Bridge Motor Driver	\$35
2	Li-Po Battery Pack	\$20
1	Power Supply Unit	\$50
		Total \$390

Table 4: Initial Estimate Budget

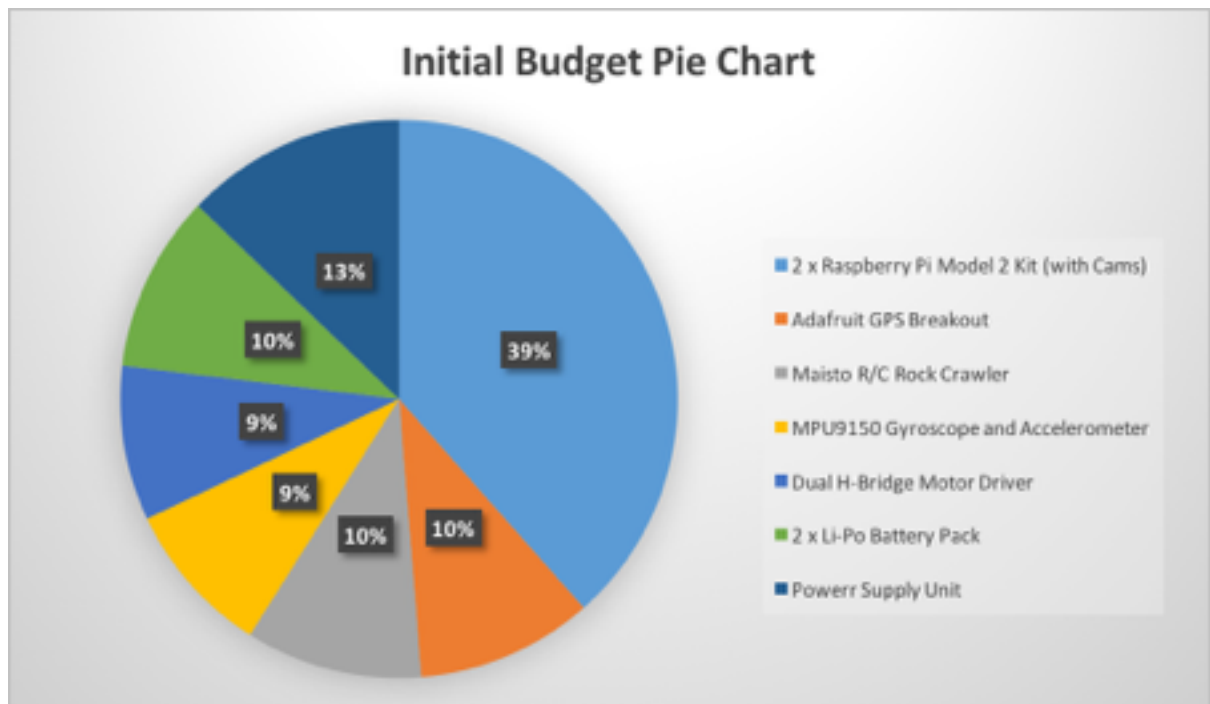


Table 5: Initial Budget Pie Chart

3. Technical description of work performed

3.1 Robot Skeleton and Motor Transmission

3.1.1 RC Car

The project was based on a robot which should be able to move and communicate. The group's choice for the base was to use an RC model car. Since the project was on a scale down version, the RC car does not need to be as big as a real tractor. The group selected an average sized RC car which is 12.5 x 7 x 8.5 inches, shown in figure 2. It has three motors, two of them are responsible for driving the front and rear wheels and the third motor is used to control the steering of the car. The driving motors are located at front and back while the steering motor is located at the back of the car. The car steers using the back two wheels. There is a platform between the front and back wheels which is able to hold the circuit and devices for the project. The team also designed a protect case to cover all the elements onto the platform.



Figure 2. RC Car skeleton

To drive and control the RC car, a microprocessor is unable to drive a motor directly by an output signal because there is very limited current that can be drawn from the processor. For this consideration, a motor driver is needed between the microprocessor and the real motor to adjust the current level.

3.1.2 Motor Driver

The team selected L298 H-Bridge Dual Bidirectional motor driver. The reason for using this motor driver is that H-Bridge is a popular circuit which allows user to easily and independently control two motors of up to 2A each in both directions. It is a motor driver ideal for robotic applications and well suited for connection to a microprocessor. The circuit

incorporates 4 direction LEDs (2 per motor), a heat sink, screw-terminals, as well as eight EMF-protection diodes. Two high-power current sense resistors are also incorporated which allow monitoring of the current drawn on each motor through the microprocessor. The power supply for this motor driver could be 6-35V DC and the team decided to use 9V DC due to the power distribution module constructed.

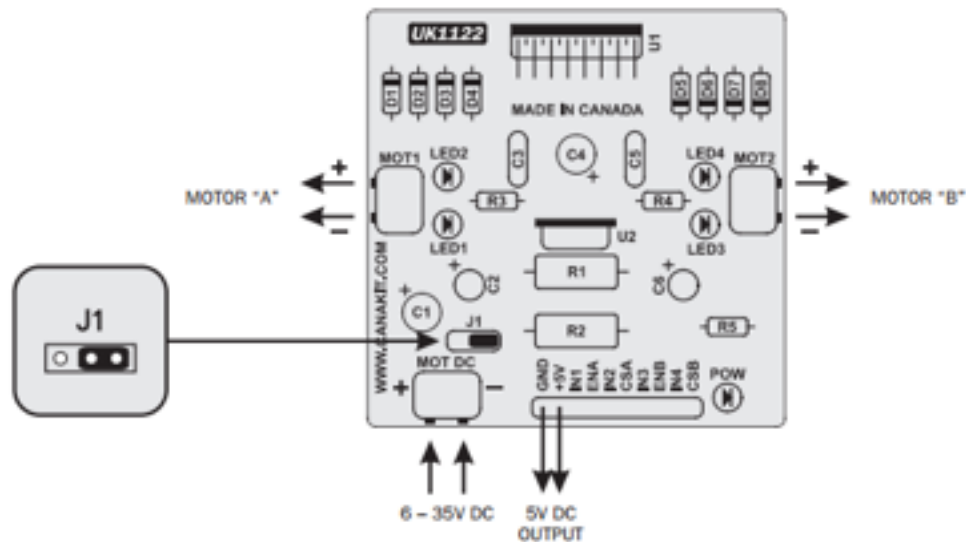


Figure 3. L-298 Motor Driver

This picture shows some RC parts of the motor driver circuit and the general input & output pins of a single L298 motor driver. Besides the power supply pins and ground pins, there are 8 inputs and 4 output pins. The output pins are located at the red and yellow LEDs while the input pins are at the bottom connectors. Each motor is controlled by setting ENA/ENB to high for the corresponding motor. Then, IN1&IN2 and IN3&IN4 can be set to high to control each motor to drive both directions. To brake the motor, IN1&IN2 or IN3&IN4 should be set to both high or low with ENA/ENB at a high signal. The following is a detailed truth table showing how the motor drivers work with different combinations of inputs.

Note: 5V DC OUTPUT is not used in this project.

ENA	IN1	IN2	ENB	IN3	IN4	Motor A	Motor B	Car
0	X	X	0	X	X	stopped	stopped	stopped
1	0	0	1	0	0	hold	hold	hold
1	1	0	1	1	0	forward	forward	forward
1	0	1	1	0	1	back	back	back
1	1	1	1	1	1	stopped	stopped	stopped
1	1	0	1	0	0	forward	stopped	forward
1	0	1	1	0	0	back	stopped	back
1	0	0	1	1	0	stopped	forward	forward
1	0	0	1	0	1	stopped	back	back

Table 6. Combinations of input

Car Modes

Forward Case:

Both motors go forward OR One motor goes forward and one motor stops

Backward Case:

Both motors go backward OR

One motor goes backward and one motor stops

Stop Case:

Both motors stop (enable is 0)

3.2 Sensor Integration

3.2.1 Sensor Introduction

3.2.1.1 Navigation sensors (IMU)

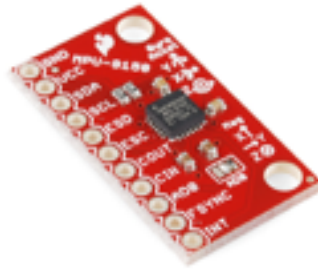


Figure 4. IMU-9150

The inertial measurement unit (IMU) chosen by the team is the Ivensence MPU-9150 Motion Tracking device with breakout board designed by sparkfun. The MPU-9150 is a multi-chip module which combines a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer. A Digital Motion Processor is also included in the hardware settings of this chip. The biggest advantage of this DMP hardware accelerator engine is such that it is capable of producing a single data stream in a combination of all the acceleration and rotational motion as well as heading information. It is more cost effective compared to process with discrete accelerometer, gyroscope and digital compass algorithms. This chip has now been widely used in motion-enabled game, navigation and 3D remote controls for Internet connected DTVs and set top boxes.

Another merit for this device is that it is designed for interface with external digital sensors, for example, a barometer or pressure sensor. This auxiliary function would be essential for the future development of the AgBOT functionalities where a down-pressure measurement is required. For configuring and reading from the external sensors: wire the Pin 7 (ES_CL) from the IMU to the SCL pin of the additional sensor and wire the Pin 6 (ES_DA) from the IMU to the SDA pin of the additional sensor. Those two pins are internally connected to an Interface Bypass Mux which is able to select the input source and then bypass to the Master I2C interface of the sensor.

The MPU-9150 consists of two dies combined into a single chip. One die houses a MPU-6050 for the accelerometer and the gyroscope. The other die houses a 8973 digital magnetometer produced by Asahi Kasei Microdevices Corporation.

3.2.1.2 Accelerometer and Gyroscope (MPU-6050)

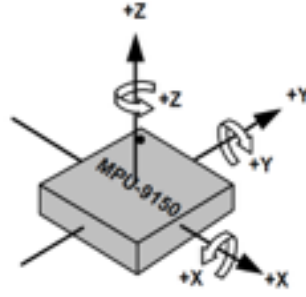


Figure 5. Graphical representation of Gyroscope

A digital accelerometer is a device that measures acceleration as a change in capacitance and converts that tiny capacitance into a digitized voltage signal and output to the external system processor via slave I2C serial interface. The Z-direction of the accelerometer is defined by a constant gravitational pull of 1g (9.8 m/s²) downwards.

Suppose an object is moving along a vector, $\vec{V}$, at specific direction. The rotation angle between that vector and X-axis, Y-axis and Z-axis are denoted by θ_{xr} , θ_{yr} and θ_{zr} respectively. And assume the projection of the vector $\vec{V}$ on the three Cartesian axes to be V_x , V_y , and V_z . The following relationships could be derived:

$$\begin{aligned}\cos(\theta_{xr}) &= \frac{V_x}{|\vec{V}|} \\ \cos(\theta_{yr}) &= \frac{V_y}{|\vec{V}|} \\ \cos(\theta_{zr}) &= \frac{V_z}{|\vec{V}|}\end{aligned}$$

According to the Pythagorean theorem, we have:

$$|\vec{V}| = \sqrt{V_x^2 + V_y^2 + V_z^2}$$

Hence, the rotation angles could be obtained by using the inverse cosine function:

$$\begin{aligned}\theta_{xr} &= \cos^{-1}(V_x/|\vec{V}|) \\ \theta_{yr} &= \cos^{-1}(V_y/|\vec{V}|) \\ \theta_{zr} &= \cos^{-1}(V_z/|\vec{V}|)\end{aligned}$$

A gyroscope is a device that measures angular rates around one or more axes. The gyroscope will not work properly with the lack of the accelerometer, therefore, the conditioning circuit employed for a gyroscope is always linked with the sensing circuitry of the accelerometer. The combination of these two inertial measurement sensors will offer the ability to capture and to track movements in a three-dimensional space. The working principle of a gyroscope is that whenever an external angular rate is applied, this will give

rise to a Coriolis force, which acts in the direction normal to the balance movement of masses inside the gyroscope.

Then, the sensing masses will convert this displacement to the magnitude of the angular rate applied. Same as the digital accelerometer, the displacement mentioned above is measured by the change in capacitance between rotors (moving electrodes) and stators (fixed electrodes), thus transforming the angular rate input into an electrical output.

$$F_c = -2m\Omega \times v$$

The above is the equation for Coriolis force where m is the mass, Ω is the angular velocity and v is the velocity of the particle with respect to the rotating system.

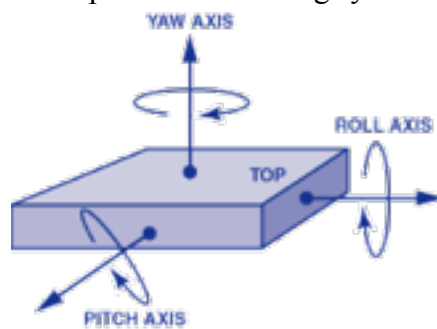


Figure 6. Gyroscope

Some people may question that if the three angle of inclination (ϕ , θ , and ψ) can be obtained directly from an accelerometer, why a gyroscope is still needed. That is because the accelerometer readings are not accurate enough to be trusted. The accelerometer is highly sensitive to surrounding noises, such as small vibrations, while the gyroscope is less sensitive to linear mechanical movement because it only deals with the change in rotation. The average of the data comes from the accelerometer and the gyroscope will give better estimation of current orientation and smooth out the noise and other fluctuation redundancies.

3.2.1.3 Magnetometer (AK8973)

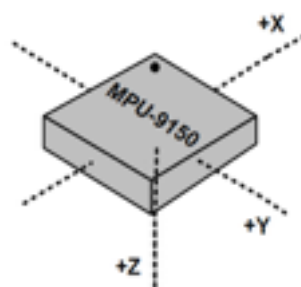


Figure 7. Graphical Explanation of Magnetometer

The AK8975 3-axis magnetometer/digital compass is connected to the MPU-9150 through the Auxiliary I2C Bus. A magnetometer is a device that capable of measuring the strength variation or the direction of magnetic field. This product uses primarily in measuring the earth's magnetic field. It is based on the idea that suppose a bar of high susceptibility ferromagnetic metal is wrapped by a coil of wire and an electrical current is applied. The bar will become magnetized and generates its own magnetic field around it. Another bar is positioned parallel to it and wrapped by the coil in reversed direction, the magnetic field generated by the second bar will cancel the first one out due to the opposite orientation. When an external magnetic field is applied, the balance of the magnetic field created by these two parallel rods would be destroyed, and the field in one core will be larger than the other. If a voltage measurement is now taken, it will detect an induced voltage proportional to the strength of the external field.

The magnetic field [Am^{-1}] is orthogonal to the direction of the current flow and to the radial vector .

$$H = i/(2\pi r)$$

The current density of can be calculated using Maxwell's equation of electromagnetism as listed below:

$$J = \nabla \times H$$

$$\nabla = (\hat{x} \frac{\partial}{\partial x}, \hat{y} \frac{\partial}{\partial y}, \hat{z} \frac{\partial}{\partial z}).$$

$$\nabla = (\hat{x} \frac{\partial}{\partial x}, \hat{y} \frac{\partial}{\partial y}, \hat{z} \frac{\partial}{\partial z}).$$

Where ∇ is the vector differential operator.

By connecting multiple paralleled bars, aligning them with X, Y, and Z axes, it is possible to measure magnetic field in three dimensions.

➤ Adafruit Ultimate GPS Breakout

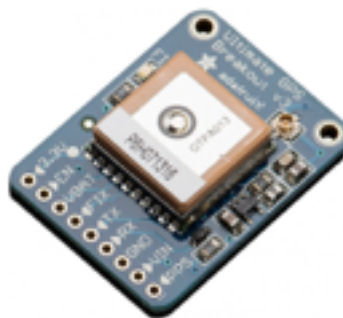


Figure. 8 GPS breakout

The Adafruit Ultimate GPS Breakout is built around the third generation MTK3339-based module. The MediaTek new generation MT3339 GPS chipset provided with the industry's highest level of sensitivity (-165dBm) and instant Time-to-First Fix (TTFF) with lowest power consumption for precise GPS signal processing. It has a 10Hz update rate and can track up to 22 satellites on 66 channels. An external antenna can also be added to the chipset with the supported embedded function for enhanced signal power and range. The red LED indicator on the breakout helps the user to visualize the acquisition status: blinking at 1 Hz while it is searching for valid fix signal and blinking once every 15 seconds when a fix has been found. This GPS module would maintain a stable connection only when there is no obstacles in the line of sight. Whenever it acquires a FIX signal, it will have the course, longitude, latitude and altitude data updated at user defined frequency.

Data collection sensors

- Raspberry Pi Camera Module



Figure. 9 Raspberry pi camera

For monitoring the back and front view of the robot, one or more cameras should be implemented on the car. The team chose to use Raspberry Pi camera module which is perfectly compatible with the raspberry. It can be directly plug into the camera interface on raspberry pi without extra connections using other pins. This camera is able to take up to 5MP photo and 1080p video at 30fps. It may not be the best resolution for a camera at the same cost but it is the most efficient one for the project considering the compatibility. Moreover, it is easy to program. The raspberry pi IDE allows the user to adjust the resolution, brightness, video length and other parameters when using the camera by some easy to understand codes.

3.2.2 Hardware (Wiring Method)

To setup a complete AgBOT, all the sensors and devices had to be connected together with the microprocessor to form a complete robotic system. Raspberry pi worked as a central processor of the robot in this project. Different from the raspberry pi camera, there was no designed connectors for devices like motor drivers, GPS chip and magnetometer on the

raspberry pi. However, there were 40 General Purpose Input Output pins on the other side of raspberry pi board. The picture below shows the detail distribution of these 40 GPIOs. The most number of pins, which are in green color, are the basic GPIO pins. These pins can be set to both input or output pins and they are able to receive or output a 0V (low) or 3.3V (high) signal. The black pins are ground pins as usual. It is convenient to connect different devices to different ground pins. Pin 1, 2, 4, 17 are the DC voltage output pins which can supply 3.3V and 5V DC to other devices connect to the microprocessor. The remaining pins are for some special usages. Pin 3, 5 are the I2C pins and Pin 8, 10 are the UART pins. I2C and UART are two popular transmission methods. They transmit data in digital bits. Both of them are useful in this project because data needs to be transmitted to the raspberry pi from the real sensors which take in analog inputs. The pins in blue and yellow color SPD and ID pins which are not used in the project.



Figure 10. Raspberry GPIO pinout

As mentioned in the previous part 3.1, motor drivers, GPS breakout and Accelerometer & Magnetometer are the devices need to be connected with the raspberry pi. The motor drivers are connected with the GPIO pins while the GPS breakout connects with the processor through the UART pins. As well the accelerometer & magnetometer requires I2C connections. Since there are three motors on the car and each motor needs 3 input pins, a total number of 9 GPIO pins have to be connected and controlled. For the UART and I2C transmission pins, only pin 3, 5 and 8, 10 can be used. So the final design of connections between devices and raspberry pi is as the following picture. In the picture, motor driver 1 is responsible for controlling the 2 driving motors and motor driver 2 is controlling the single steering motor. Moreover, GPIO pins 16 and 18 are connected to a speed control circuit which is a simple MUX. Two bits are able to form four selections 00, 01, 10, 11 and each of them leads to a different supply voltage to the motor driver so the motor's speed can be controlled by the output of pin 16 and 18.

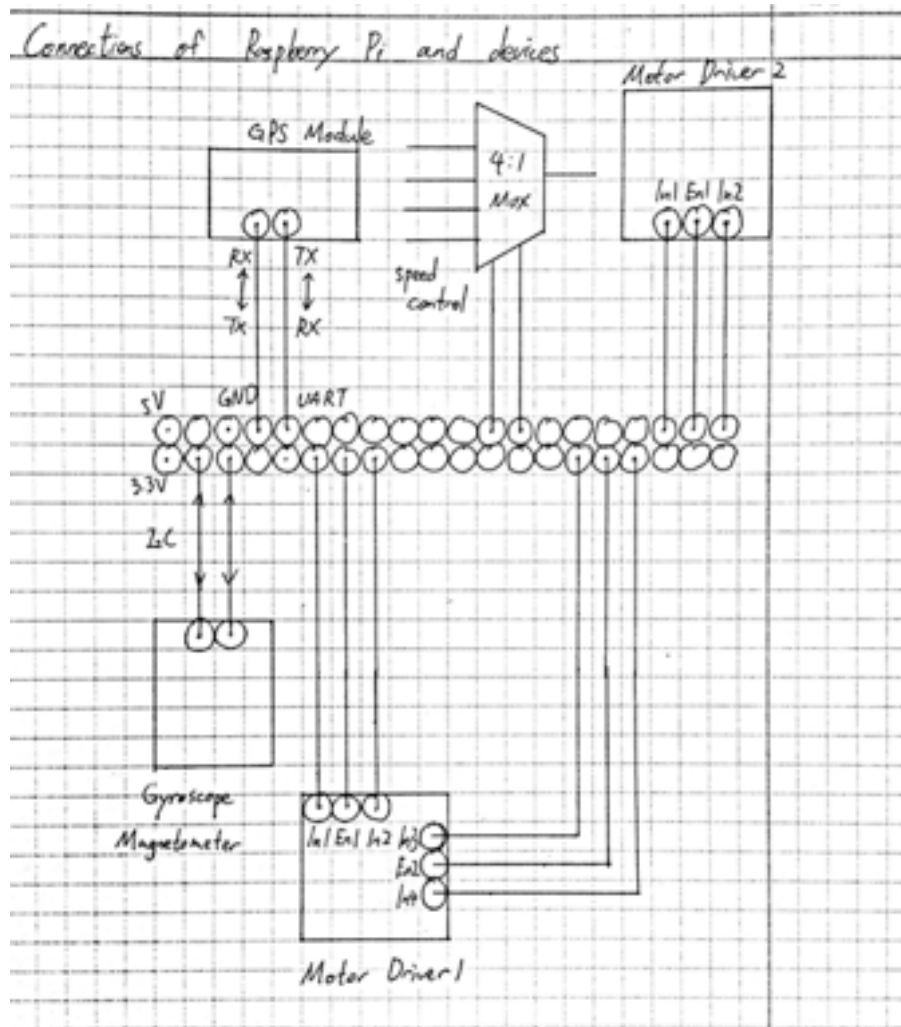


Figure 11. Wiring method of integrated sensor system

3.2.3 Software (Library Data Association)

While the previous part introduced the physical connections between each sensor and the Raspberry Pi, it is also important to know the programming theory behind the operation of the sensors involved.

Raspberry Pi, at the heart of the system, has a Linux operating system which is used to run the software algorithm code in C++ language. The group uses this microprocessor to interface with the sensors and upload the data and image files to the internet. The Inertial Measurement Unit (MPU-9150) and the GPS module are attached to the GPIO pins of the Raspberry Pi via I2C and UART respectively.

➤ Installing WiringPi

WiringPi is a powerful C++ library designed for the Raspberry Pi that includes a wide variety of GPIO tools. It can manipulate the GPIO pins of the Ras-Pi (the 40 input-output pins on the

board) and contains functions like Serial Communication, I2C communication, PWM (Pulse-Width Modulation), and many others. To install WiringPi, go to the WiringPi website and download the latest version of the code. Once downloaded, extract the .tar.gz file to a local directory and then run the extracted folder to make the WiringPi library installed on the RasPi.

➤ Interfacing the MPU-9150

As described earlier in the hardware session, a valid I2C connection requires two wires: a SCL wire and a SDA wire. SCL is the clock line which is used to synchronize all the data transfers. By setting the clock speed (up to 100 KHz), user is able to get data read/write at the desired frequency. The SDA data usually transmit in sequence of 8 bits with the last bit defining the reading/reading behavior on those addresses. There is an open-source driver written by Jeff Rowberg and was ported to Raspberry Pi by Richard Christ available on the GitHub repository. With this I2C for accessing the MPU-9510, it is possible to read the raw data from accelerometer, gyroscope and magnetometer or use the digital motion processor which performs 6-axis sensor fusion. The DMP also has the ability to calculate yaw, pitch and roll angles in three-dimensional manipulation.

The following instructions were used for preparing the I2C connection on Raspberry Pi:

```
$ sudo apt-get update
$ sudo apt-get install i2c-tools libi2c-dev

$ sudo nano /etc/modules
- At Add the following lines to the bottom of this file
- Add "snd-bcm2835"
- Add "i2c-bcm2708"
- Add "i2c-dev"

$ sudo nano /boot/config.txt
- Add the following lines to the bottom of this file
- Add "dtparam=i2c1=on"
- Add "dtparam=i2c_arm=on"

$ git clone git://github.com/richardghirst/PiBits.git
$ cd PiBits/MPU6050-Pi-Demo
$ sudo apt-get install libgkmm-3.0-dev
$ nano I2Cdev.cpp
- Change all occurrences of "/dev/i2c-0" to "/dev/i2c-1"
- Save file
$ nano setup-i2c.sh
- Change all occurrences of "/dev/i2c-0" to "/dev/i2c-1"
- Save file
$ make
$ ./setup-i2c.sh
$ sudo i2cdetect -y 1
- the IMU should use address 68
```

➤ Interfacing the GPS Unit

The Adafruit Ultimate GPS Breakout has to be connected with raspberry pi using UART transmission. The UART transmission has two pins which are transmitter (Tx) and receiver (Rx) on both the GPS chip and the raspberry pi. The transmitter of one device should be connected to the receiver of the other one. This forms an effective UART transmission. The

GPS sends standard NMEA messages at a rate of 10 Hz when a GPS lock is maintained. The group has set the baud rate for serial interface to be 9600, which is good enough for this project and power saving. The data the GPS chip receives from the satellites are called ONEM data. It is a special form of analog data which can represent the longitude, latitude and altitude of the current location. There are several modes of geographical coordinate representation available simultaneously. Those data are continuously pushed to the serial bus and then parsed to the desired output format (GPGGA) the user programmed. The group uses Decimal Degrees (DD) for latitudes and longitudes (51.499400, 0.122759) other than the Degrees Minutes Seconds (DMS) coordinate format which has been used most frequently in car navigation system. Since the Adafruit GPS library is mainly written for Arduino which has an .ino file incompatible with the Raspberry Pi, the group decided to use an open-source GPS library developed by Walter Dal Mut on the GitHub repository. It is very simple to use: First, download the libgps-master package and compile the source code on the target. Second, obtain libgps.a and include <gps.h> header file in the desired application. To print the latitude and longitude data on the terminal, following codes could be added to the main function in the application:

```
gps_init(); //initialize the communication
loc_t data;

while(1) {
    gps_location(&data); //expose data from GPS
    printf("%lf %lf\n", data.latitude, data.longitude);
}
```

3.3 Wireless Communication Setup

Since the project has a requirement on bi-directional data transmission between an external workstation and the robot, the group has to set up a communication interface in between these two devices. Also, to make the robot stand alone and to enhance the mobility, the team need to free any means of wire attachment made from the outside environment. For this reason, a Raspberry Pi Wi-Fi module must be used even though the Ethernet cable would also allow the group to have internet access. The group uses a router as the network creator. The router is capable of providing a range up to 100 meters for the use of monitoring and controlling. The robot and any device that the user tends to use as a base station would be connected as long as they are in the same network that the router created. Another advantage for using a router is that it can easily hold a stable network whether there is a valid internet access or not. Under most situation, when the robot is operating in an open environment where a Wi-Fi signal might be too weak to connect, the operator only needs to plug the router to any power outlet and let the raspberry pi search and connect to it.

The method of connecting the Raspberry Pi using a 802.11b/g/n Wi-Fi dongle is straightforward. First, edit the raspi-config panel and select boot to Desktop with default

login. Second, click on the Wi-Fi Config icon on the desktop or run the command “wpa_gui” if it is missing. Last, click “scan” in the window popped out and choose the network. If it requires a password, enter it in the PSK field.

Another important functionality the group achieved is remotely control the desktop interface of the Raspberry Pi from other computer. This function requires the use of VNC (Virtual Network Computing) software, which shares the Raspberry Pi desktop to the screen of workstation, and transmits the keyboard and mouse from the controller over the network from the remote host. The following instructions explained the process of installing VNC Server on the Raspberry Pi and make it running at startup:

```
$ sudo apt-get install tightvncserver
- Confirm installation by typing "Y"
$ vncserver :1
- Enter and confirm a password
- type "no"
$ cd home/pi
$ cd .config
$ mkdir autostart
- Make a new directory inside .config
$ cd autostart
$ nano tightvnc.desktop
- Edit the contents of the file with the following text
- [Desktop Entry]
- Type=Application
- Name=TightVNC
- Exec=vncserver :1
- StartupNotify=false
- Save the changes and exit
```

3.4 Displaying live video feed

Real-time video feed is one of the major functions of the base station user interface. This section will focus on the RasPi setup and library preparation for a successful video streaming. More detailed code for HTML web interface will be explained later in Section 4.4.2 Base Station User Interface. For live streaming of the video from the Raspberry Pi camera module, the group installed the MJPG-streamer library and Apache web server with PHP support on the Raspberry Pi. The user can type the following bash commands to install the dependencies:

```
$ sudo apt-get install libv4l-dev libjpeg8-dev subversion imagemagick libapache2-mod-php5 php5 apache2
$ svn co https://svn.code.sf.net/p/mjpg-streamer/code/MJPG-streamer
$ cd MJPG-streamer/ jpg-streamer
$ sudo make USE_LIBV4L2 = true
```

The last command will build the MJPG-streamer folder on the Raspberry Pi. After making the file, some executables will be generated in the same folder. The folder should contain the following files: mjpg_streamer, input_uvc.so, input_file.so and output_http.so. These files would allow the MJPG-streamer to work successfully. Instead of uploading all the camera

captured data in a video format, the streaming server needs a sequence of JPEG files to stream. Because the load required to generate the JPEG is much less than in other image formats. A build-in command line application for RasPi camera module can be used for convenience. The “`raspistill`” command will take a picture from the camera module and it has been designed to meet various specifications. The basic scripting for this part is to make the capture behavior run in background and to output the images to designed directory which can be referred from other places. The following example can be used in terminal:

```
$ mkdir /home/pi/stream
$ raspistill -w 640 -h 480 -q 5 -o /home/ pi/stream/pic.jpg -tl
100 -t 99999999 -th 0: 0: 0 -n &

$ LD_LIBRARY_PATH =/home/pi/MJPEG-streamer/mjpg-streamer/ /home/
pi/MJPEG-streamer/mjpg-streamer/mjpg_streamer -i "/home/pi/MJP
G-streamer/mjpg-streamer/input_file.so -f /home/ pi/stream -n
pic.jpg" -o "/home/pi/MJPEG-streamer/mjpg-streamer/output_htt
p.so -p 8081 -w /home/pi/MJPEG-streamer/mjpg-streamer/ www" &
```

By running the above commands, the Raspberry Pi will write the JPEG images from camera at a rate of 10 per second and running in the background. The streaming server is also start to work. The only thing left is open a web browser, type in the current IP address and the video port defined in the codes above. For example: `http://<IP ADDRESS>:8081`.

4. Test data with proof of functional design

4.1 Hardware design efforts

For the entire robotic system, the team has to meet several design specifications. For hardware, an independent power system, an integrated sensor system and a wireless communication system are three initial requirements. An independent power system is aimed to supply power to all the devices in the system including the motors of RC car, the microprocessor and the sensors in the system. An integrated sensor system is expected to wire all the sensors together with the microprocessor to form a data transmission and computing system. An integrated sensor system means that the parts are no longer working separately but as a whole system which can decide solutions. A wireless communication is another essential part for the robot. The wireless communication system should allow the robot with the power and sensor system to transmit data bidirectionally with the base station in a range of 100 meters. For the software part, a set of codes in raspberry needs to form a complete algorithm to enable the robot to work according to the inputs taking from the sensors. That is to navigate the robot according to the GPS coordinates given. Moreover, the base station UI also requires software efforts. Users should be able to view the processed data and video feedback in the base station.

4.1.1 Power Distribution Module

The RC car for this project required a lot of different components. The raspberry-pi, the various different sensors, front and back cameras, motors, GPS and accelerometer to name a few. To power all these different elements, we needed to come up with some sort of a power module that would provide us with steady currents and voltages to power these elements. So the team decided to build a power distribution module. There were three different voltages that we needed to provide; they were:

- 1) 3.3V for the sensors and GPS
- 2) 5V for the raspberry-pi
- 3) 9V for the DC motors

After looking at the specifications we had to meet, the best way to meet the requirements was to make a power module that would be providing us the three voltages from the main battery. To start off the design procedure, we first decided to replicate the model present in our lab notes:

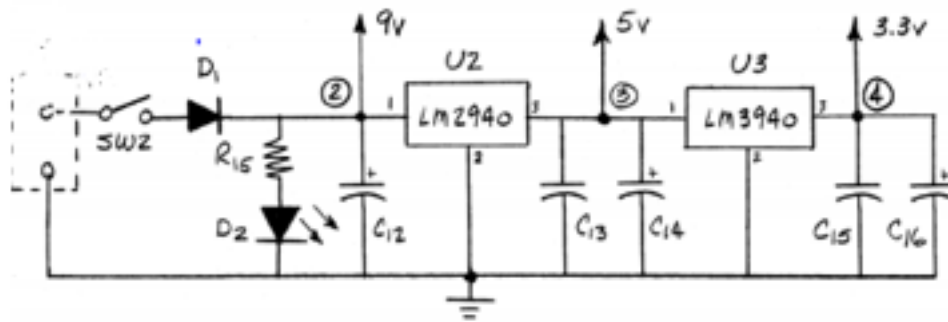


Figure 12. Voltage Regulator Circuit

The above model uses two different voltage regulators, the LM2940 and LM3940. The module above looks to be a great simple solution to our problem. Although it does provide us with the right amount of voltage, it was not a good solution to meet our specification. This was because although we were getting the right voltage, the current level in our circuit was not enough. DC motors usually require higher amounts of current to start. But, the current provided by our module was very low ($\sim 0.5 - 0.7\text{A}$) as compared to the current that was required to startup the motors ($\sim 1 - 1.2\text{A}$). Therefore, the motor would not start if we did not provide it with some sort of torque (e.g. push the tyres). One way to solve this was to use different voltage regulators that are rated at higher current.

The first choice of replacement were linear voltage regulators, LM7805, LM7809. The circuit for the power is shown in fig X and Fig Y. This were able to provide stable regulated power up to certain time. But due to high internal heating, the circuit does into thermal shutdown causing entire system to shut OFF, as a precautionary step, and thus allowing only 5 minute of run before shutting the system OFF.

To resolve the heating problem, team decided to switch the voltage regulators from simple linear regulators to switching regulators. The reason why switching regulators will work better than linear regulator is because a switching regulator works by taking small chunks of energy from the input source and transmit it to the output, thus keeping the average power consumption low. Since the power dissipation is low, internal temperature will be lower than the one created in linear regulator. [4] The switching regulators, used to drive the voltages, are shown in table X below:

Voltage Regulator	Regulated Output
LM2734	9 V at 1 A
TPS62110	9 V at 1.5 A
TPS54232	9V at 2 A
LM2575	5V at 1 A
LM1117	3.3V at 800mA

Table 6. Voltage Regulators with ratings

Full schematic of the circuit is provided in APPENDIX 3.X along with the PCB design.

4.1.2 Robot Protection case/Base

There are a lot of different components that need to be on the robot as it is running. The two raspberry-pi's, GPS, accelerometer, motor drivers, camera, batteries and as well as the power distribution module itself. As there are so many components that need to be present on the robot; and as the robot does not have a predefined base to carry all these components, we had to come up with a base for all these components. We wanted the base to be big enough to be able to contain all the components while not hampering with the movement of the robot. As the base had to be big, we had to make sure that it did not come into contact with the tires of the robot while it is running. Keeping that in mind, the team decided to make a two-part base. The first part would elevate the second part of the base, while the second part of the base would contain all the different components required. To design the base, we used the NX10.0 software.

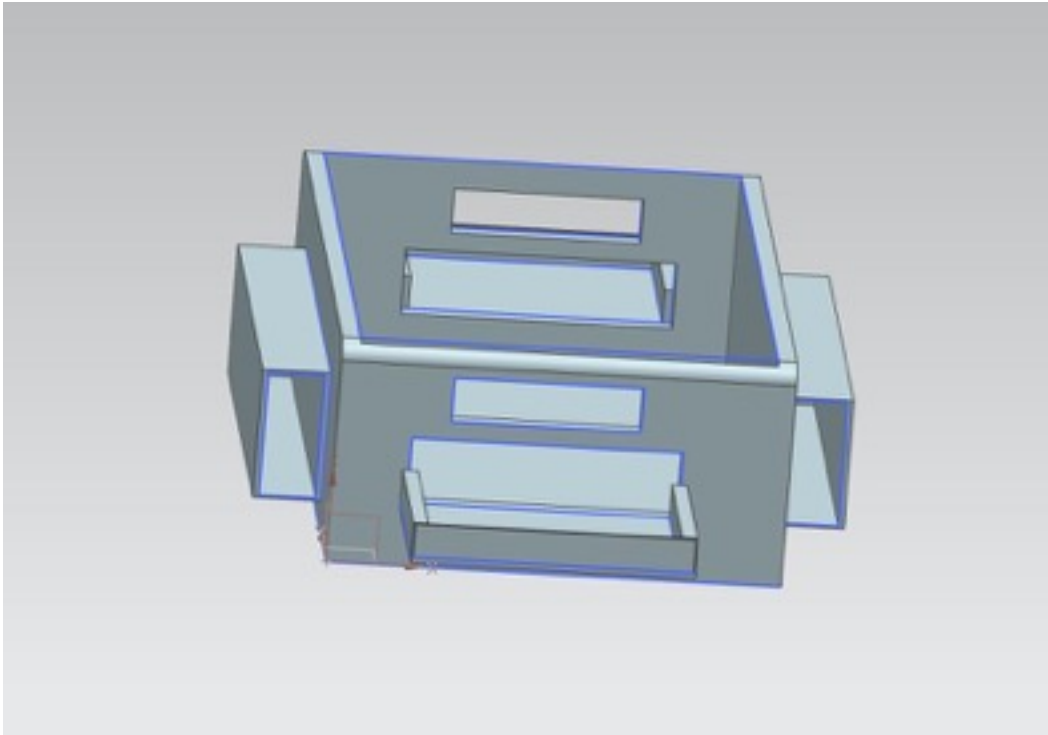


Figure. 13 Upper Case

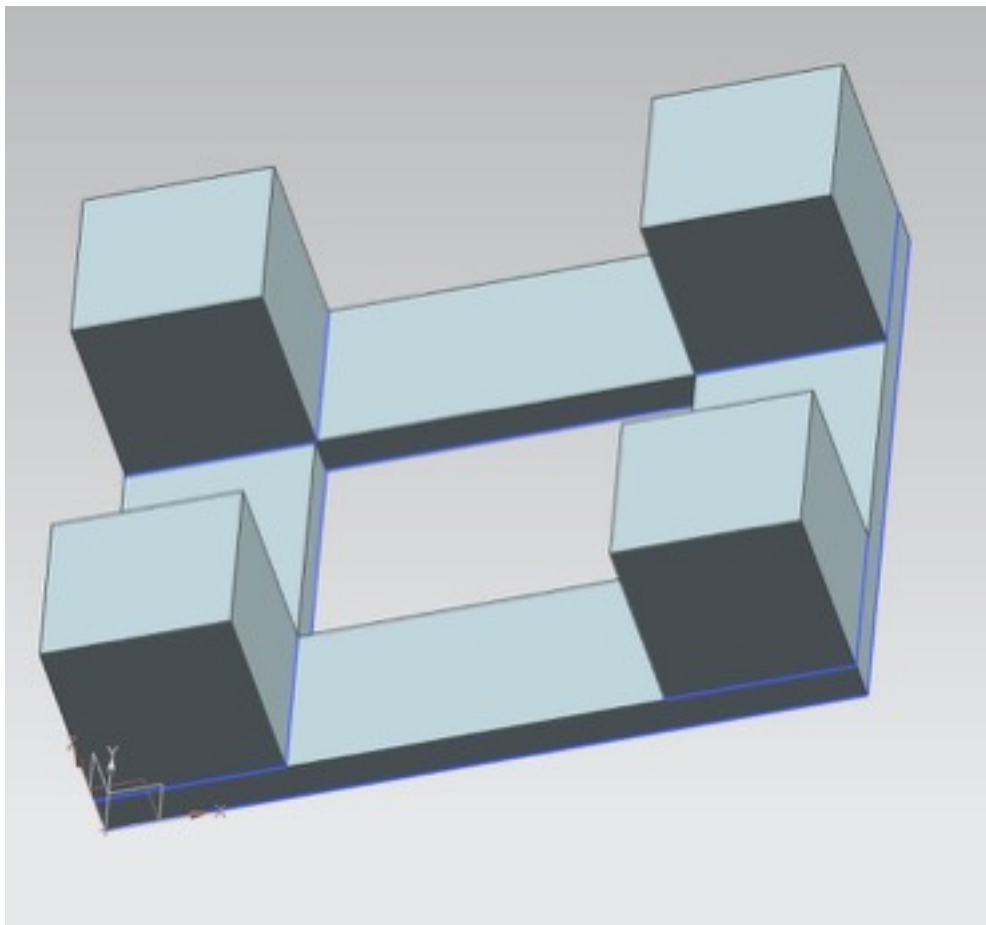


Figure 14. Lower Case

After designing the base on NX10.0, the team decided to take the help of the DECS office in order to 3-D print our bases. Although DECS was able to print our lower base, the

upper base was way too large for the machine to 3-D print. Thus, we had to scrap the design and come up with a new one. For the new base, the team agreed on using a simpler design. By using a plastic board as our upper base and then using Velcro to stick the different elements on to it was the easiest and quickest engineering solution that the team could think of and we decided to go ahead with it.

4.2 Hardware implementation

The main source of power for the robot comes from 2 Li-Po Battery capable of supplying 11.1 V individually. The two battery pack will be connected in parallel combination thus keeping the voltage constant at 11.1 V. This battery pack will be then connected to voltage regulators to produce the regulated power that will be used to power different elements. The internal connections between Raspberry Pi and sensors are shown in Fig X on page X. The working chart for the entire system is shown in figure X below.

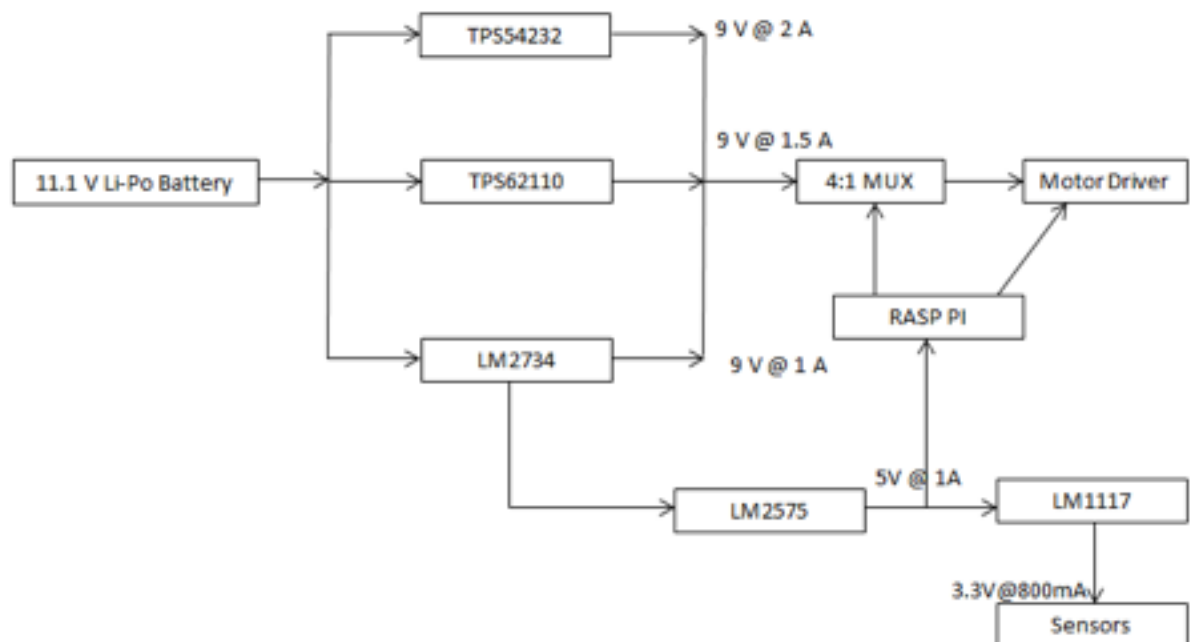


Figure 15. Hardware Connection

The first prototype was constructed on breadboard and thus it was congested with wires going across the breadboard, making it impossible to find any wrong connection and troubleshooting. Thus to simplify the layout of the circuitry, team decided to create PCB with

internal connection, thus eliminating any use of wire between any elements. The full schematic of the PCB with the board layout is shown in APPENDIX 3.2.

4.3 Software and interface design requirements

4.3.1 Raspberry Pi System Software

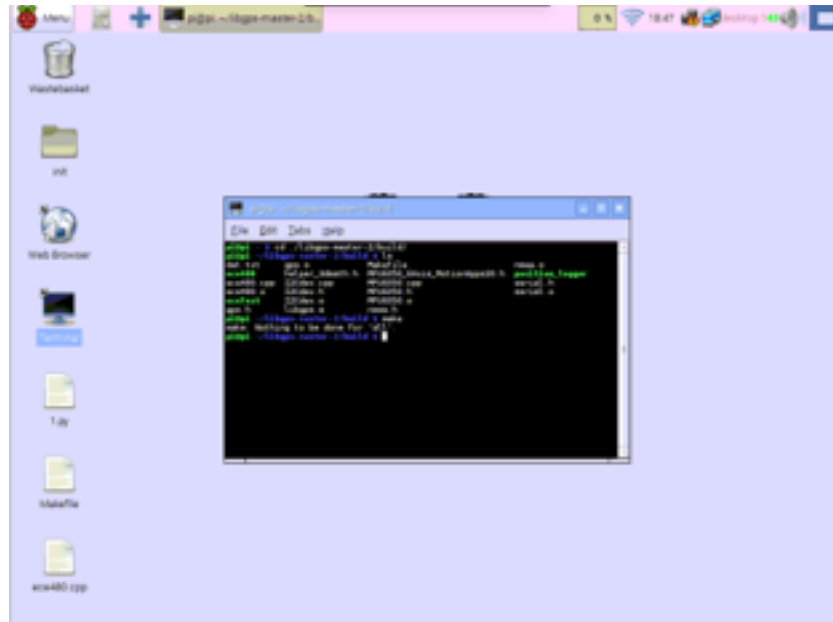


Figure 16. Raspberry pi system

Raspberry Pi is based on Linux operating system. It has built in software including file system, web browser, terminal, text editor and all other operating system have. This project requires skills using Linux OS, command line, C++, html and Php. Since the robot will be working under a non-monitoring situation, the robot has to have the ability of operating and managing all task by itself. It also has to record all instructions and operations in memory. The robot should also be able to work continually for 5-8 hours per day.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Rasp! U!</title>
<link rel="stylesheet" type="text/css" id="theme" href="css/index.css">
</head>

<body>

<center>
<div id="connect">

</div>

<div id="controls">




</div>

<script type="text/javascript" src="js/jquery.min.js"></script>
<script type="text/javascript" src="js/index.js"></script>

<center>
<div>

</div>

<div id="streamdata" style="float: none;">

</div>
</center>
</body>
</html>
```

Figure 17. Code of Web Server

4.3.2 Local Web Server

The robot works as a server on the local network as the basic part of data communication. This functionality requires database and data transmission fundamental technology of building a server by using a microcontroller. Basically it has to have an open port and request handler to receive data, process data and send data out. So other devices in this local network can connect with the robot by ssh or the prepared web page by IP address. The ssh method lets remote command line login can be achieved and provides full file access for all remote users. The prepared web page is a limited public source that all device in the network can read. All available user activities are provided by the web page programmer.

4.3.3 User Interface Design

The user interface is one important part of this project. It is an interface that lets users view the robot status and streamed video, and also control the robot. The user interface requires video streaming which is read-only content. The web page actually prints an image in the view and updates the image 30 frames per second. The image is generated by the code of camera center and stored on the disk so the web page can read it. The robot status is also read-only. It reads a txt file and prints lines on the web page. The txt file is modified by the main control code. The control part is write-only; user can send instruction by tapping the button, and the tapping event will trigger a script file, then the robot will run the script file as a primary action to finish the user's instruction.

4.4 Software implementation

4.4.1 GPS-Based Autonomous Navigation Algorithm

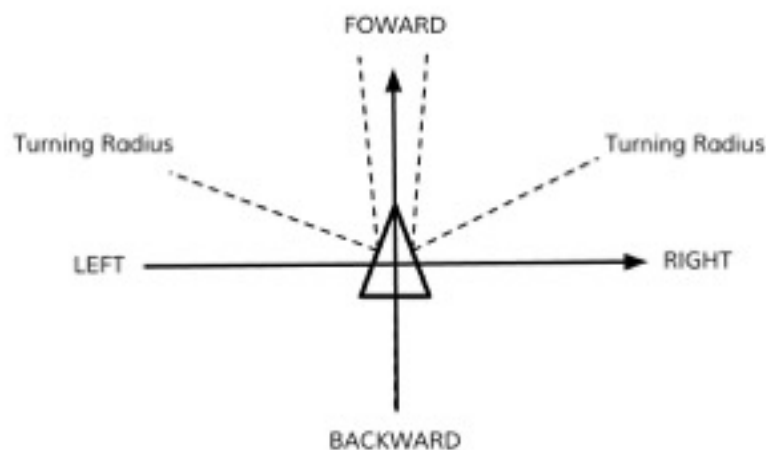


Figure 18. Navigating Algorithm

The system reference frame is calculated by the forward moving direction of the robot. This frame can be divided down into five separate zones. Zone 0 segregated Zone 1 and Zone 2 by allowing the degree of uncertainty for any particular heading. Zone 3 and Zone 4, ranged by

two obtuse angles below, are segregated by the left and right turning radius. The turning radius defines the smallest circular turn that the vehicle is capable of turning. In this case, it is the steering limit the vehicle can achieve without moving backwards. Any GPS waypoint falls in different zones will trigger different microprocessor commands to the motor. For example, a GPS location assigned in zone 1 will lead the robot to turn left and drive forward. However, a GPS location assigned in zone 4 will invoke a backtracking algorithm for the moment, since the location is beyond the angle of steering radius.

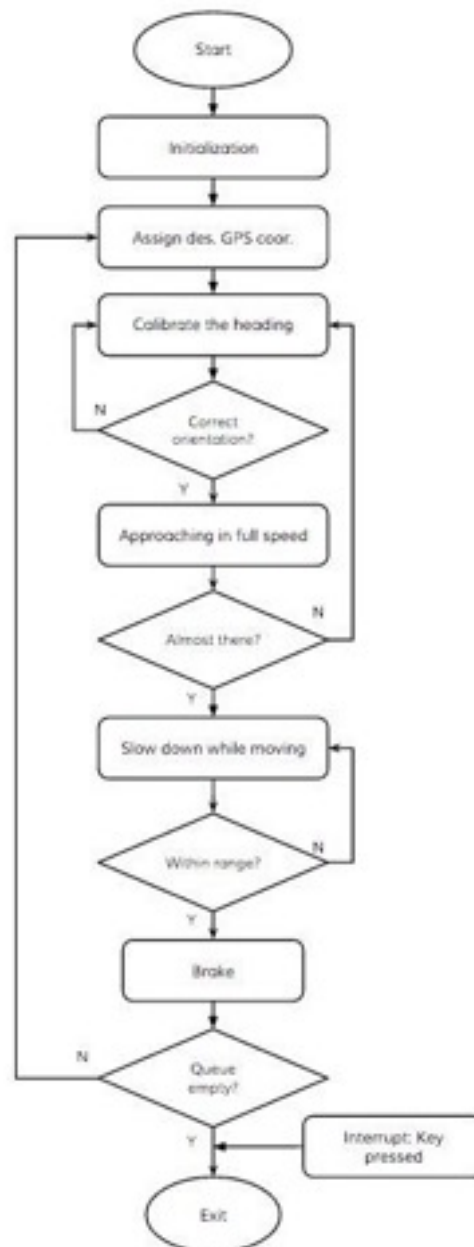


Figure 19. Flow-chart of Algorithm

Code for setting program constants:

```
////////////////////////////////////code constants////////////////////////////////////
float acceptErrorAngleNeg = -10.0;
float acceptErrorAnglePos = 10.0;

float leftErrorAngle = -70.0;
float rightErrorAngle = 70.0;

float slowingDownDistance = 10.0;
float stopDistance = 3.0;

#define pi 3.14159265358979323846
////////////////////////////////////code constants////////////////////////////////////
```

These constants are preset before any lines of the code. These parameters set the basic line of how user want the robot behave. The acceptable error angle is used for decreasing wiggling from error data input, it can also increase the moving speed of the robot. Left and right error angle are according to the minimum turning radius of the robot. So if the destination is out of the turning range, the robot will do special instruction to achieve the goal. Slowing down distance sets the safety distance between robot and destination to avoid error or crushing.

Code for initializing:

```
printf("start to set up");
wiringPiSetupGpio();

pinMode(17, OUTPUT);
digitalWrite(17, 0);
pinMode(22, OUTPUT);
digitalWrite(22, 0);
pinMode(27, OUTPUT);
digitalWrite(27, 1); // enable

pinMode(5, OUTPUT);
digitalWrite(5, 0);
pinMode(13, OUTPUT);
digitalWrite(13, 0);
pinMode(6, OUTPUT);
digitalWrite(6, 1); // enable

pinMode(16, OUTPUT);
digitalWrite(16, 0);
pinMode(21, OUTPUT);
digitalWrite(21, 0);
pinMode(20, OUTPUT);
digitalWrite(20, 1); // enable

// Speed pins enable
pinMode(23, OUTPUT);
pinMode(24, OUTPUT);
// Preset the speed to REGULAR
digitalWrite(23, 1);
digitalWrite(24, 1);

// pre-defined GPS destination coordinates
float des_latitude = 42.723349;
float des_longitude = -84.488728;

// define current loc for testing use (please comment them out when running)
float current_latitude = 42.733359;
float current_longitude = -84.588828;

//gps_init();
//loc_t data;

float gpsNewLat = 0;
float gpsNewLng = 0;

mpu.initialize(); // Initialize the OMP in MPU9150
devStatus = mpu.dmpInitialize();
```

These lines of codes first initialize the GPIO ports, including pin number and input/output mode. Second part of the code initializes the destination information and current location for next calculations. The third part initializes the GPS and Gyroscope.

Code for calculating distance between two annotations:

```
float distance(float lat1, float lon1, float lat2, float lon2, char unit) {
    float theta, dist;
    theta = lon1 - lon2;
    dist = sin(deg2rad(lat1)) * sin(deg2rad(lat2)) + cos(deg2rad(lat1)) * cos(deg2rad(lat2)) * cos(deg2rad(theta));
    dist = acos(dist);
    dist = rad2deg(dist);
    dist = dist * 60 * 1.1515;
    switch(unit) {
        case 'M':
            break;
        case 'K':
            dist = dist * 1.609344 * 1000;
            break;
        case 'N':
            dist = dist * 0.8684;
            break;
    }
    return (dist);
}
```

This part of code calculates distance between two geo locations. It takes latitude and longitude of two locations, do the math and return the distance.

Code for sending GPIO instructions:

```
void instructionLoop(float angleDif, float distanceDif) {
    if (distanceDif < slowingDownDistance) {
        //do slow motor down instruction
        digitalWrite(23, 0);
        digitalWrite(24, 1);
    }
    else {
        //do regular speed
        digitalWrite(23, 1);
        digitalWrite(24, 1);
    }
    if (distanceDif < stopDistance) {
        //front motor
        digitalWrite(17, 0);
        digitalWrite(22, 0);
        //back motor
        digitalWrite(5, 0);
        digitalWrite(13, 0);

        //left front
        digitalWrite(16, 0);
        digitalWrite(21, 0);

        printf("in range stop");
    }
    else if (angleDif > acceptErrorAngleNeg && angleDif < acceptErrorAnglePos) { //forward
        //front motor
        digitalWrite(17, 1);
        digitalWrite(22, 0);
        //back motor
        digitalWrite(5, 1);
        digitalWrite(13, 0);

        printf("forward only");
    }
    else if (angleDif > leftErrorAngle && angleDif < acceptErrorAngleNeg) { //left front
        //left front
        digitalWrite(16, 1);
        digitalWrite(21, 0);
    }
}
```

This part of code takes angle difference of the local frame and the world frame. It then allocates the comparing destination heading direction on the local frame and select GPIO instructions to send to pins.

Code for comparing local frame and world frame:

```
current_latitude = data.latitude;
current_longitude = data.longitude;

float calAngle = atan((des_longitude - current_longitude)/(des_latitude - current_latitude));
float curAngle = loop();
if (curAngle > -180.0 && curAngle < 180.0) {
    float angleDifference = curAngle; //- (90.0 - calAngle);
    float distanceDifference = distance(des_latitude, des_longitude, current_latitude, current_longitude, 'K');
    printf("degree: %7.2f\n", angleDifference);
    instructionLoop(angleDifference, distanceDifference);
}
```

This part of code shows how the angle difference has been calculated. It first translates geo location of the destination to the comparing angle of the world frame. Then it processes the current heading angle by the data from gyroscope. Finally it calculates the difference of these two angle to get the angle difference that instruction loop used.

Code for recording data to txt file:

```
//Writing the data to a text file
ofstream myfile;
myfile.open("dat.txt"); //Opening a file with filename "dat.txt"
myfile << current_longitude << endl; //Exporting the data to the file
myfile << current_latitude << endl;
myfile << data.altitude << endl;
myfile << data.speed << endl << curAngle << endl << angleDifference << endl;
myfile.close(); // Closing the file

loop();
```

This part of code outputs the current robot status into a txt file. The file is used by user interface to print all sensor information on web page.

4.4.2 Base Station User Interface (Human-machine Interfaces)

Live Video feed and remote interface base

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Raspi UI</title>
  <link rel="stylesheet" type="text/css" id="theme" href="css/index.css">
</head>
<body>
  <center>
    <div id="connect">
      <a onclick="GoTo('ON')">Connect to Engine</a><br>
    </div>
    <div id="controls">   </div>
  </center>
  <script type="text/javascript" src="js/jquery.min.js"></script>
  <script type="text/javascript" src="js/index.js"></script>
  <center>
    <div>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <a> </a><br>
      <div id="streamdata" style="float: none;">
        
      </div>
    </div>
  </center>
</body>
</html>
```

This is a html code for index of the web. It connect the initialization button to the php file of action processing. It place the control buttons and add javascript on it for user interrupt actions. It also enable and set the video streaming block on the page.

Remote Control Panel

```
<?php
$action = $_POST['GO'];
switch ($action)
{
  case "ON":
    shell_exec('sudo /home/pi/remoteRobot/controller/init.sh');
    break;
  case "FORWARD":
    shell_exec('sudo /home/pi/remoteRobot/controller/forward.sh');
    break;
  case "BACKWARD":
    shell_exec('sudo /home/pi/remoteRobot/controller/backward.sh');
    break;
  case "LEFT":
    shell_exec('sudo /home/pi/remoteRobot/controller/left.sh');
    break;
  case "RIGHT":
    shell_exec('sudo /home/pi/remoteRobot/controller/right.sh');
    break;
}
echo json_encode(array('success'=>1,'direction'=>$action));
?>
```

This is the php code for action selection. It takes the instruction from the user interface and do the switch and finally run the *sh* file of the instruction.

Data Reporting

```
<!DOCTYPE html>
<html>
<head>
  <title></title>
</head>
<body>

<div>
  <?php
    echo file_get_contents('dataStored.txt');
  ?>
</div>

</body>
</html>
```

This is the php code for printing txt file on the webpage. It will display the robot status and other hardware information on the user interface.

4.6 Problem encountered and solution

Team 8 was able to perform most of the activities without running into major problem. There was only one problem that team faced during the entirety of the project. Since the team was using linear voltage regulators to supply power to all elements. Since the requirement current was more than 1 Amps for the motor driver and RaspBerry PI, energy loss in the voltage regulator will be around 120 WATTS [3] and as a result the internal temperature will rise. This will force the regulator to go into thermal shutdown and will stop providing power to the elements, proving only five minute of run time.

Team 8 solved the problem by changing the voltage regulator for linear regulator type to more efficient voltage regulators called switching regulator. Since the switching regulator transfer the power in short amount within the IC, it produces less of the heat loss and thus keep the regulator from shutting it OFF. [5]

5. Integrated System Test

5.1 Test Run 1

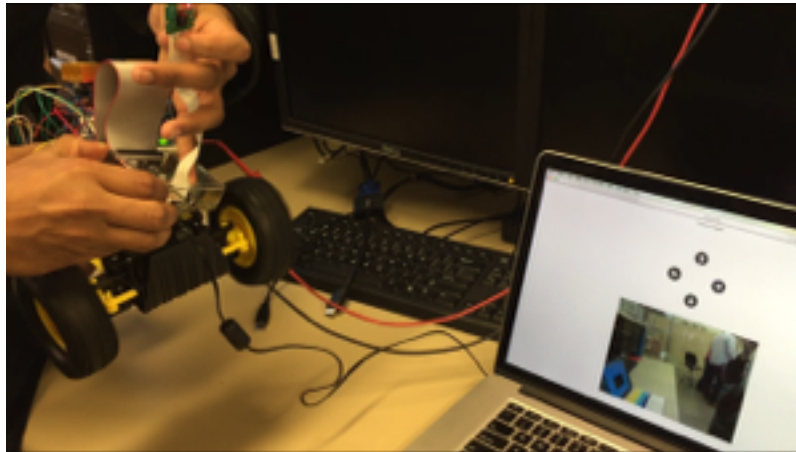


Fig 21. UI Live Video Feed with Remote Control

The team did the first test just after all the parts arrived. The car was briefly constructed using the breadboard and two motor drivers. The raspberry pi was used to given inputs to the motor drivers in order to control the car. A camera was also implemented in order to record video. The power supply for the all the stuff in this test was from the ECE 480 lab. An initial version of control loop code was created. In addition, an user interface was established using a website address. There were four buttons indicating forward, backward, left and right on top of the website. There was also an area for live video streaming. This test was fully successful. The car could be controlled to move in specific directions by the button pressed on the website. The video taking from the camera could also be well displayed on the website. This test demonstrated the first prototype of the project.

5.2 Test Run 2



Fig 22 . GPS Data Obtained on Raspberry Pi Desktop

The second test was performed in the outdoor environment. This test is aimed to test the GPS signal receptivity and its compatibility with Raspberry Pi by showing the acquired location data on the Raspberry Pi terminal. The Raspberry Pi was powered by a small Ni-Mh battery pack from 3.3 volts and has been boosted up to supply 5 volts voltage by using a voltage booster. The group operated this test on the path between engineering building and communication arts building. The FIX signal acquired successfully in several minutes after booted up. The obtained longitude, latitude, and altitude data with their errors are displayed on the terminal via VNC Remote Desktop software and being updating while the position changes.

5.3 Test Run 3

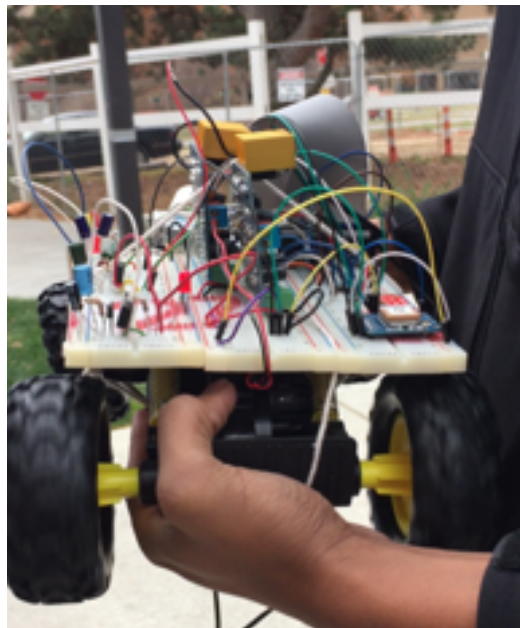


Fig 23. Integrated system test on a breadboard

In the first test run, the team connected all the devices on a breadboard. The team also used a substitution battery as the power supply because the breadboard was unable to run under high current level. The team went outdoor, around 0.05 miles from Engineering Building. The first step of the test was to setup the router. It was plugged into one of the outlet of engineering building near the exit. Secondly, the raspberry pi and the integrated sensor system were boost up by the connection with the battery through a voltage regulator circuit. Then the raspberry pi was able to connect with local wireless network created by the router. The steps above were successful as expected. Since the power was supplied to the whole system, the GPS breakout and IMU started to work. It took roughly five minutes for the GPS breakout to get a fix signal. After that, the GPS breakout was able to send the location data to the raspberry pi. The data was viewed in the software as expected as well as the IMU data. For this first test, the team set a fixed GPS coordinate in advance and tried to let the robot go to that destination. The robot was going forward, backward several times and the power supply went down at this moment. The team tried to reconnect the power supply to the robot but it did not

work. After a few seconds, the whole system shut off and some parts of the circuit were overheated. So the first test was stopped here.

In the first test, the basic functions of the system were tested. But there were still problems in the power supply and software algorithm. The power supply was able to boost up the whole system for about 1-2 minutes but some heating problems occurred after that period of time. The total energy stored in the substitution battery was also not enough to support a long time test. For the algorithm, the robot was doing some judgement according to its location but it was not enough for to reach the final destination without human intervention. The team decided to work further on these two parts for the next test.

5.4 Test Run 4

The group ran the same previous teste, but instead of suing the breadboard, the team used the PCB. The robot performed as expected and did not have any difficulties in performing the actions that it had in the previous tests.

6 .Design Issues

6.1 Product Liability & Safety

The liability of the product should be discussed through hardware and software parts. The independent power supply system may have some concerns. It supplies power to multiple devices integrated on the robot. The total current and power consumption were large and this causes thermal shut down within the voltage regulator. In several tests, the voltage regulators got overheated which resulted in the shutdown of entire system. In order to compensate the overheating problem, team decided to switch the voltage regulator from linear to switching voltage regulator thus improving the efficiency, since switching regulators are power efficient than linear regulators. However, the liability could still be improved by using minimal noise induced elements.

6.2 Limitation of Design

There were several limitations of the project due to the budget and technical issues. First of all, the transmission range of the robot system was limited to a range of maximum Wi-Fi extension which is 100 meters. The actual AgBOT competition requires the robot to plant rows of half mile long. Since the BATS transmission system was not used in this project, there was no way to extend the transmission range to that distance. Secondly, the lifetime of the power supply was also limited. Supporting a whole system, the battery could last for no longer than 10 minutes, which is not sufficient enough for the group needs to simulate the whole seeding process. Last of but not least, the size of the robot itself is a challenge. The real AgBOT will have a base of an agricultural size vehicle which does not run on DC motors.

6.3 Future Improvement

The design team is looking forward to future improvements of the project. Some other sensors could be implemented on the robot to collect more data for the user to better monitor and analyze the robot. A speed sensor and a temperature sensor may be the most possible solution for this improvement. A speed sensor can let the user know not only which direction the robot is going but how fast it is going along a row. This could help the user to improve seeding accuracy. As well, a temperature sensor could give the user more information about the environment. Other possible improvements of the project are: Using more powerful transmission system,

7. Final cost, summary, schedule and conclusions

7.1 Final cost

After testing out our design and then transferring all the components from the breadboard to the PCB and testing it, the team was confident that we would not need anymore components to buy or have any more expenses. Out of the \$500 that was given to us in our budget, we were left with \$40 of our budget.

Quantity	Name of the item	Cost/Unit (\$)
2	Raspberry Pi Model 2 Kit (with Cameras)	\$75
1	Adafruit Ultimate GPS Breakout	\$40
1	Maisto R/C Rock Crawler	\$40
1	MPU9150 Gyroscope and Accelerometer	\$35
1	Dual H-Bridge Motor Driver	\$35
2	Li-Po Battery Pack	\$20
1	Power Supply Unit	\$50
1	Printed Circuit Board	70
Total		\$460

Table 6. Final Cost

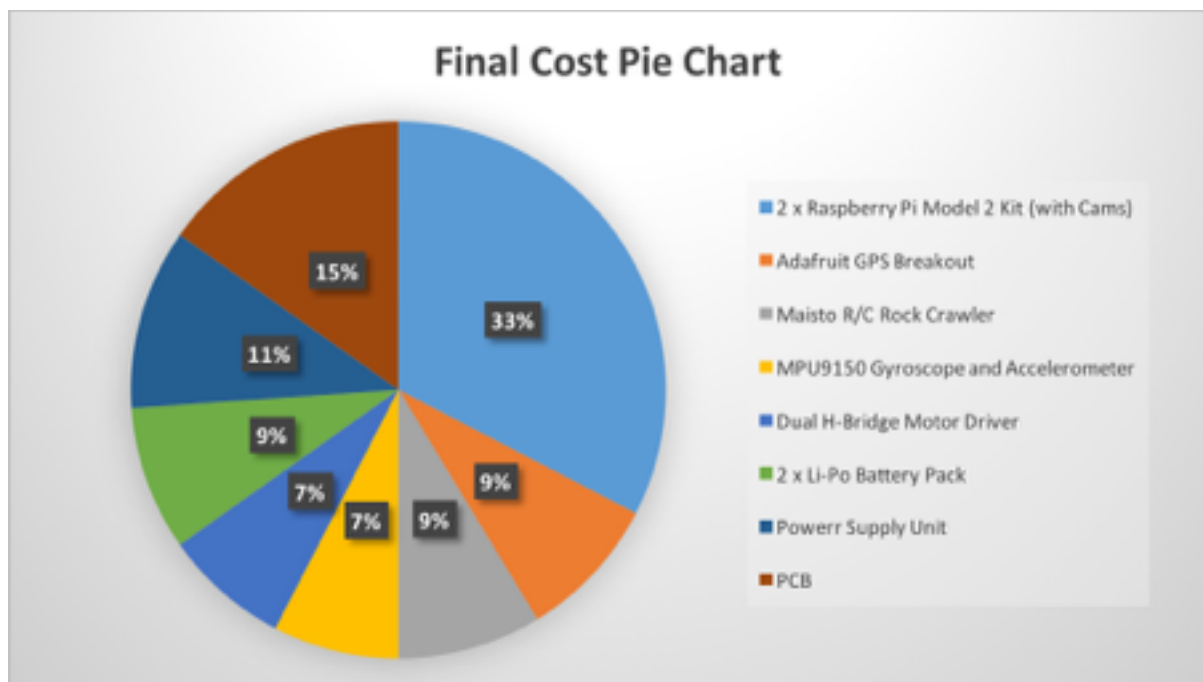


Table 7. Final Cost Pie Chart

7.2 Summary

Team 8 was able to complete the requirements efficiently in accordance with the design specifications of the project. The group was successful in designing a robot that would autonomously make 12 straight rows and provide front and back live video streaming. The team also created a User Interface that has the capability of displaying the information of the robot, including the live video feed of the robot. The robot is also capable of being remotely controlled from the base station in case the user wants to change the course of action of the robot. The robot uses GPS and accelerometer to align itself and go to the predefined positions existing in the program. The team was also able to come up with a battery life indicator, which was not a requirement, which tells the remaining running time/battery time. However, due to time limitations and also budget factors, the team was unable to do extensive testing and also was unable to produce the base that the team desired. The team believes that it made significant progress in terms of meeting the client requirements.

Although the robot is capable of meeting the design specifications, there are some categories where it falls a little short. One of the main shortcomings of the design is the range the robot can operate in. As the team is using Wi-fi as the communication means, the range the robot can operate in is only about 100 meters. If the BATS system was available, the team is confident the range could be increased to around 700 meters. Another setback of this project was the size of the robot. Because of the low budget, the team was unable to use a full size tractor, which would be ideal for an AgBot. Moreover, as the robot is small in size, it is unable to perform the seeding and fertilizing functions. While these failures are clear setbacks in the project, the team is confident that given more budget and resources, they can tackle these problems with relative ease.

If the team were to continue working on this project, there are a lot of other features the team could add on to the existing features and also enhance the performance of the existing features. One thing that the team would like to enhance on this project is the range of the robot and also the running time.

7.3 Conclusion

The team was successful in building a robot that runs autonomously and meets all the design specifications provided. The robot is capable of recording and transmitting front and back live video stream, running autonomously based on GPS coordinates, indicating the battery life present, being controlled remotely and also be controlled through a smartphone. This gives the user great ease and control. The team is very happy with the efforts every team member did on this project and is confident that it has met with all the different client requirements.

8. Appendix

8.1 Individual Efforts

Yirui Wu



Technical Role: She has research experience on vision based control system using Raspberry Pi, which is mainly focused on the embedded system programming of a remote-controlled camera in C++ to regulate image capturing and processing. Therefore, she is proficient with the Raspberry Pi operating system, programming environment and other hardwares, such as IMU, that work hand in hand with the Raspberry Pi. She is mainly dealing with the Raspberry Pi programming interface establishment and how it interacts with other electronic devices in this project. She made major decisions on what specific type of design components the group is going to use in this project. For example, the Sparkfun MPU-9150 breakout is one of the chips she has encountered before in her research area. She has read the manufacture manuals thoroughly to understand the internal configuration of each sensor and exploited all the available functions the sensor provide. One of the direct result is to decide which pin is going to connect with the Raspberry Pi and to write values into the sensor register in order to select appropriate function. In the software algorithm the sensor setup and initialization are reflected by the installation of necessary libraries and the compilation of all the sensor source files into an executable application. Hence, she ensured the effective communication between the Raspberry Pi and the desired sensor by calling developed functions in our project algorithm script. She is also incharge of the wireless communication setup and wrote the script for the Raspberry Pi camera module to create the live video feed, which is done by using a third-party streaming server and make the camera module running in background.

Non-Technical Role: Yirui Wu as the presentation coordinator for this project. She is the primary designer for all the three presentation PowerPoints as well as the poster. She is also instrumental in writing other technical documents such as the pre-proposal, proposal and the final report.

Harsh Desai



Technical Role: Harsh Desai did primary research on the design specification and was responsible for creating initial design for the AgBOT. Due to his extensive experience in PCB designing and knowledge in the functionality of the voltage regulator, he was given the duty of creating the Power Distribution Unit and creating PCB that eliminates the use of cables in excess. Due to the different requirements for the power, Harsh Desai has to come up with the solution to provide sufficient power using different voltage regulators. There were some issues with the internal heat within the regulator, but Harsh Desai was able to solve the issue by switching the linear regulator to switching regulator. The final design of PCB was also designed by Harsh Desai using EAGLE software.

Non-Technical Role: Harsh Desai was the project manager and was responsible for staying in contact with the sponsors and facilitators. His other responsibility includes arranging meeting with sponsors and facilitator and keeping track of the team's progress by creating a Gantt chart and updating it constantly. He was the main source of communication between the team members and the facilitator. He also contributed in organizing presentations and writing proposal and final report.

Chen Hao



Technical Role: He focused on the programming part of this project. He wrote the server end code for data communication between devices and the robot, includes php, css and js codes for building a fully functioning server on the raspberry pi. He wrote the html code for user interface including control panel and status block. He is responsible for designing the layout of function blocks and what type of data is showing in the user interface. He wrote main cpp file, including control loop code and all data receiving, calculating and transmitting code for robot controlling and self-data-processing. He wrote the algorithm for calculating robot physical status and next moving instructions, which is the core code of the robot operating in programming.

Languages used: HTML, Javascript, CSS, C, C++, PHP and command line.

Software used: Coda 2, Xcode, Photoshop and terminal.

Non-Technical Role: He was the webmaster of this project. He build the team website with thousands lines of code. He was also the graphic editor of the web design. He is also instrumental in writing other technical documents such as the pre-proposal, proposal and the final report.

Sumanto Pal



Technical Role: Sumanto's main contribution was in the hardware part of the project. He was instrumental in creating and designing the power distribution module for the robot. As he is adept in integrated circuits and has taken many classes in that field, he did most of the calculations for the power distribution module; i.e. selecting the right components for the module and what their values should be. He was also instrumental in coming up with the design for the base of the robot needed for carrying the different components. He designed both the upper and lower base for the robot and also was in charge of putting it all together. His experience with the NX10.0 software and good designing skills made him a good candidate to finish designing and building the base of the robot. He also helped in building the PCB for the power module where he was influential in soldering the different parts on to the PCB. As he had previous experiences with EAGLE software, he also helped with the design of the PCB using EAGLE.

Non-technical Role: Sumanto was the lab coordinator for the project. He researched the cost of different components needed for building the project. As he was the lab coordinator he was also the person responsible for ordering all the parts and picking them up from the office. He was also resourceful in arranging lab meetings.

Yanqi Wang



Technical: He researched on how to use different devices separately and how to integrated them together on hardware in the project. He researched on how to use the raspberry pi system including the setup, wireless connection and software updating. He mainly focused on how to use the GPIO pins on the board to control other devices such as the motor driver and GPS breakout. He designed and built up a whole connection map for the raspberry pi, camera, RC car, motor driver and sensor system on the breadboard. During the procedure, he tested every device using the power supply in the lab to simulate it as the output of raspberry pi or power distribution module. He tested the motor drivers with the RC car by different combinations of inputs. He created a table on the logic of how to control the motor driver to drive the car in different directions. He also researched on how to get the data from the GPS breakout and IMU sensor. He tested the GPS breakout outdoor to ensure the GPS chip would get a fix signal. He made efforts to displayed the data collected from the GPS and IMU on the raspberry pi so it could be able to use in the software algorithm. He as well did research and gave advice to other parts of the project such as setting up router, compiling software. He was able to attend all the meetings and tests during the semester. He gave useful advices and discussed with other group members to help improve the test runs.

Non-Technical: He is the document preparation of the group. He was able to setting up the structure and shared it with the group of the documents. He also participated in creating the powerpoint of the three presentations.

Appendix 2 – Literature and website references

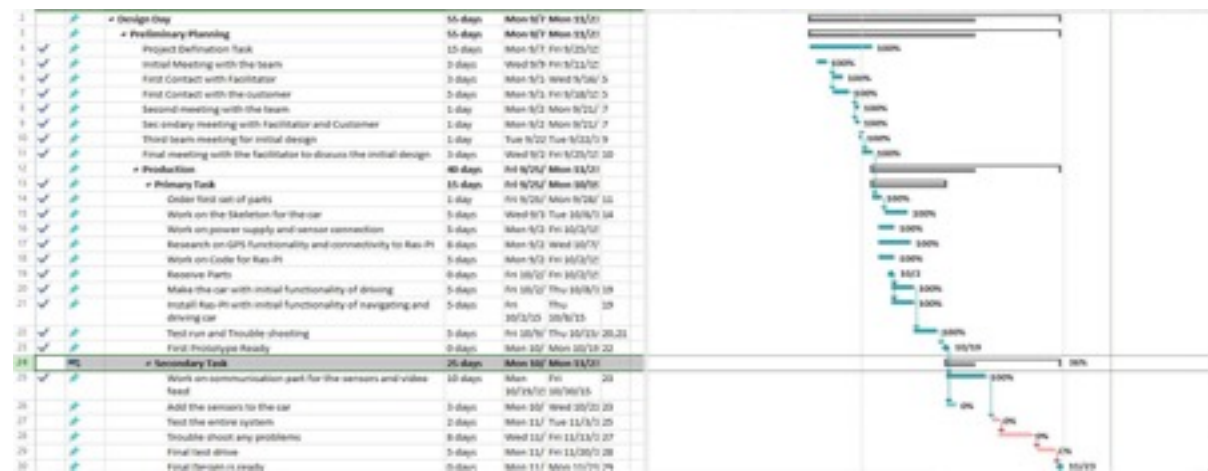
REFERENCES

1. InvenSense. MPU-9150 Datasheet, 18 Sept. 2013. Web. http://www.invensense.com/mems/gyro/documents/PS-MPU-9150A-00v4_3.pdf
2. L298 High Bridge Dual Motor Driver Datasheet. Web
<http://www.st.com/web/en/resource/technical/document/datasheet/CD00000240.pdf>
3. Simple Guide to the RPi GPIO headers and Pins, Matt, 9 June. 2012. Web.
<http://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/>
4. LM7809 Datasheet
<https://www.fairchildsemi.com/datasheets/LM/LM7809.pdf>
5. Advantage of Switching regulator over linear regulator.
<http://www.ti.com/lit/an/snva558/snva558.pdf>

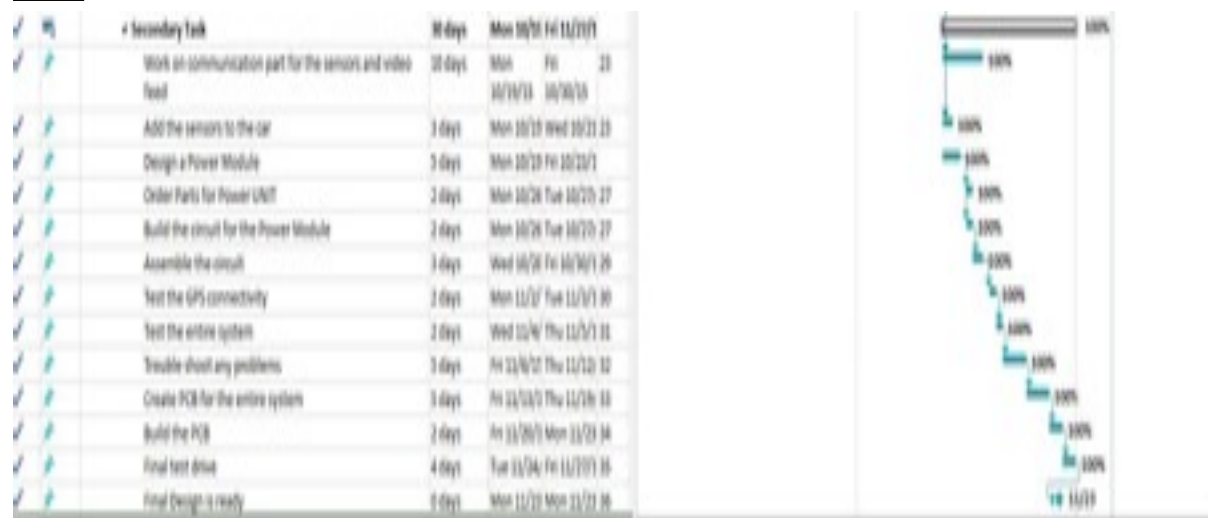
Appendix 3 and beyond – Detailed technical attachments

Gantt Chart

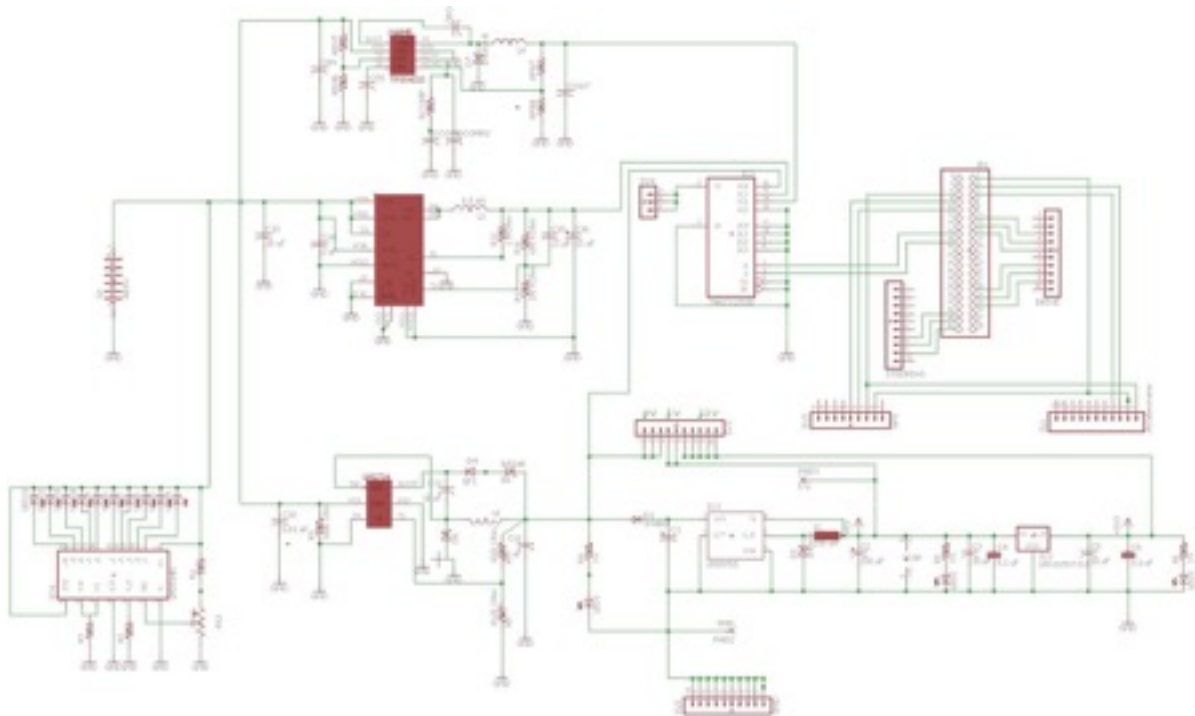
Initial



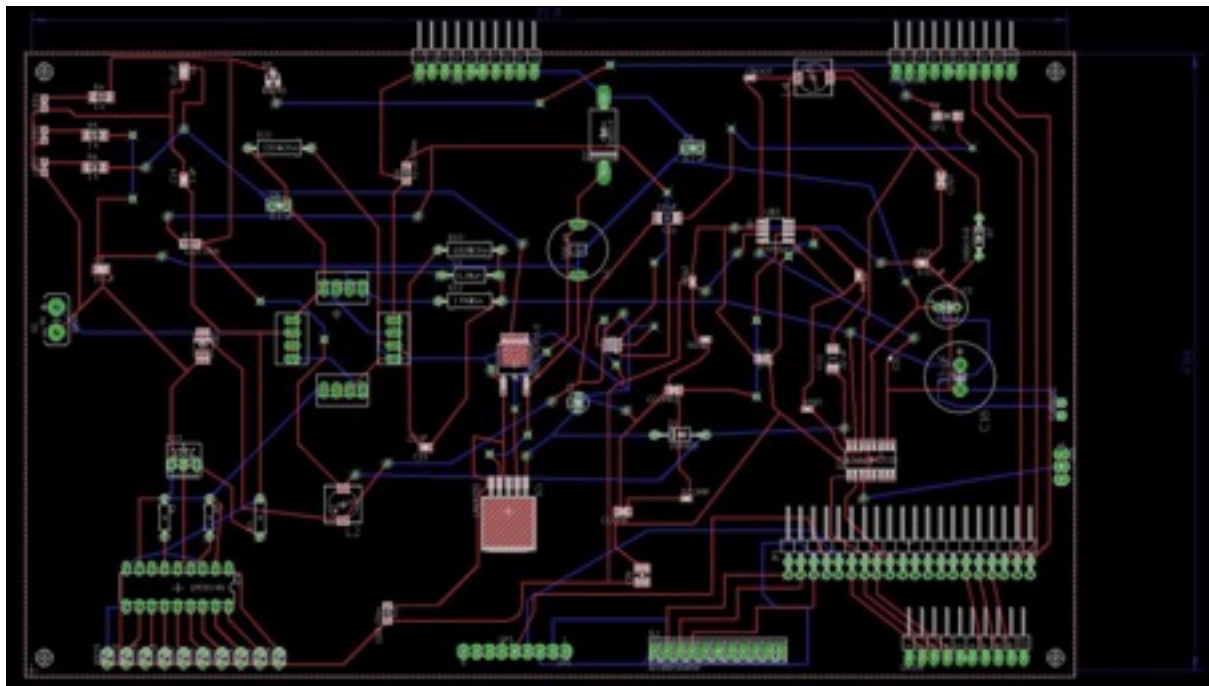
Final



PCB Schematic



PCB Layout



Project Code

ece480.cpp

```
//
// ece480.cpp
//
//
// Created by Chen Hao on 15/10/30.
// Modified by Yirui Wu on 11/18/15
// Added 3-Axis Mag feature
// Modified by Yirui Wu on 12/03/15
// Added speed control and file stream functions
//

//#include "ece480.hpp"
#include <iostream>
#include <fstream>
//#include <random>
#include <string>
//#include <vector>
//#include <cmath>
#include <stdio.h>
#include <math.h>

#include <stdint.h>
#include <unistd.h>

#include <stdlib.h>

#include <wiringPi.h>

extern "C"
{
#include <gps.h>
}

//MPU-9150 Libraries
#include "I2Cdev.h"
//#include "MPU6050.h"
#include "MPU6050_6Axis_MotionApps20.h"
//#include "helper_3dmath.h"

using namespace std;

////////////////////code
constants////////////////////////////////////
float acceptErrorAngleNeg = -10.0;
float acceptErrorAnglePos = 10.0;

float leftErrorAngle = -70.0;
float rightErrorAngle = 70.0;

float slowingDownDistance = 10.0;
float stopDistance = 3.0;

#define pi 3.14159265358979323846
```

```

////////////////////////////////////code
constants////////////////////////////////

// class default I2C address is 0x68
// specific I2C addresses may be passed as a parameter here
// AD0 low = 0x68 (default for SparkFun breakout and InvenSense evaluation board)
// AD0 high = 0x69
MPU6050 mpu;

// uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the actual
// quaternion components in a [w, x, y, z] format (not best for parsing
// on a remote host such as Processing or something though)
#define OUTPUT_READABLE_QUATERNION

// uncomment "OUTPUT_READABLE_EULER" if you want to see Euler angles
// (in degrees) calculated from the quaternions coming from the FIFO.
// Note that Euler angles suffer from gimbal lock (for more info, see
// http://en.wikipedia.org/wiki/Gimbal\_lock)
// #define OUTPUT_READABLE_EULER

// uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see the yaw/
// pitch/roll angles (in degrees) calculated from the quaternions coming
// from the FIFO. Note this also requires gravity vector calculations.
// Also note that yaw/pitch/roll angles suffer from gimbal lock (for
// more info, see: http://en.wikipedia.org/wiki/Gimbal\_lock)
#define OUTPUT_READABLE_YAWPITCHROLL

// uncomment "OUTPUT_READABLE_REALACCEL" if you want to see acceleration
// components with gravity removed. This acceleration reference frame is
// not compensated for orientation, so +X is always +X according to the
// sensor, just without the effects of gravity. If you want acceleration
// compensated for orientation, use OUTPUT_READABLE_WORLDACCEL instead.
// #define OUTPUT_READABLE_REALACCEL

// uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see acceleration
// components with gravity removed and adjusted for the world frame of
// reference (yaw is relative to initial orientation, since no magnetometer
// is present in this case). Could be quite handy in some cases.
// #define OUTPUT_READABLE_WORLDACCEL

// uncomment "OUTPUT_TEAPOT" if you want output that matches the
// format used for the InvenSense teapot demo
// #define OUTPUT_TEAPOT

// MPU control/status vars
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, != 0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// orientation/motion vars
Quaternion q; // [w, x, y, z] quaternion container
VectorInt16 aa; // [x, y, z] accel sensor measurements

```

```

VectorInt16 aaReal;      // [x, y, z]          gravity-free accel sensor measurements
VectorInt16 aaWorld;    // [x, y, z]          world-frame accel sensor measurements
VectorFloat gravity;    // [x, y, z]          gravity vector
float euler[3];         // [psi, theta, phi] Euler angle container
float ypr[3];           // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector

// packet structure for InvenSense teapot demo
uint8_t teapotPacket[14] = { '$', 0x02, 0,0, 0,0, 0,0, 0,0, 0x00, 0x00, '\r', '\n' };

float loop() {
    // if programming failed, don't try to do anything
    if (!dmpReady) return 10000.0;
    // get current FIFO count
    fifoCount = mpu.getFIFOCount();

    float returnAngle = 100000.0;

    if (fifoCount == 1024) {
        // reset so we can continue cleanly
        mpu.resetFIFO();
        printf("FIFO overflow!\n");

        // otherwise, check for DMP data ready interrupt (this should happen frequently)
    } else if (fifoCount >= 42) {
        // read a packet from FIFO
        mpu.getFIFOBytes(fifoBuffer, packetSize);

#ifdef OUTPUT_READABLE_QUATERNION
        // display quaternion values in easy matrix form: w x y z
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        //printf("quat %7.2f %7.2f %7.2f %7.2f      ", q.w,q.x,q.y,q.z);
#endif

#ifdef OUTPUT_READABLE_EULER
        // display Euler angles in degrees
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetEuler(euler, &q);
        printf("euler %7.2f %7.2f %7.2f      ", euler[0] * 180/M_PI, euler[1] * 180/M_PI,
euler[2] * 180/M_PI);
#endif

#ifdef OUTPUT_READABLE_YAWPITCHROLL
        // display Euler angles in degrees
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

        return ypr[0] * 180/M_PI;

        printf("ypr %7.2f %7.2f %7.2f      ", ypr[0] * 180/M_PI, ypr[1] * 180/M_PI, ypr[2] *
180/M_PI);
#endif

#ifdef OUTPUT_READABLE_REALACCEL
        // display real acceleration, adjusted to remove gravity
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetAccel(&aa, fifoBuffer);

```

```

        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
        printf("areal %6d %6d %6d      ", aaReal.x, aaReal.y, aaReal.z);
    #endif

    #ifndef OUTPUT_READABLE_WORLDACCEL
        // display initial world-frame acceleration, adjusted to remove gravity
        // and rotated based on known orientation from quaternion
        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetAccel(&aa, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal, &q);
        printf("aworld %6d %6d %6d      ", aaWorld.x, aaWorld.y, aaWorld.z);
    #endif

    #ifndef OUTPUT_TEAPOT
        // display quaternion values in InvenSense Teapot demo format:
        teapotPacket[2] = fifoBuffer[0];
        teapotPacket[3] = fifoBuffer[1];
        teapotPacket[4] = fifoBuffer[4];
        teapotPacket[5] = fifoBuffer[5];
        teapotPacket[6] = fifoBuffer[8];
        teapotPacket[7] = fifoBuffer[9];
        teapotPacket[8] = fifoBuffer[12];
        teapotPacket[9] = fifoBuffer[13];
        Serial.write(teapotPacket, 14);
        teapotPacket[11]++; // packetCount, loops at 0xFF on purpose
    #endif

        printf("\n");
    }
    return returnAngle;
}

// angleDiff from -180 to 180 degree
// distanceDif in meters
//Speed control MUX select bits: s0 = Pin16(GPIO23), s1 = Pin18(GPIO24)
//Define s0s1 = 00 : STOP (zero power input)
//Define s0s1 = 01 : LOW (enter low speed mode)
//Define s0s1 = 11 : REGULAR (regular speed mode)

void instructionLoop(float angleDif, float distanceDif) {
    if (distanceDif < slowingDownDistance) {
        //do slow motor down instruction
        digitalWrite(23, 0);
        digitalWrite(24, 1);
    }
    else {
        //do regular speed
        digitalWrite(23, 1);
        digitalWrite(24, 1);
    }
    if (distanceDif < stopDistance) {
        //front motor
        digitalWrite(17, 0);
        digitalWrite(22, 0);
        //back motor
        digitalWrite(5, 0);
        digitalWrite(13, 0);

        ///////////////////////////////////

```

```

        digitalWrite(16, 0);
        digitalWrite(21, 0);

        printf("in range stop");
    }
    else if (angleDif > acceptErrorAngleNeg && angleDif < acceptErrorAnglePos) { //forward
        //front motor
        digitalWrite(17, 1);
        digitalWrite(22, 0);
        //back motor
        digitalWrite(5, 1);
        digitalWrite(13, 0);

        printf("forward only");
    }
    else if (angleDif > leftErrorAngle && angleDif < acceptErrorAngleNeg) { //left front
        //front motor
        digitalWrite(17, 1);
        digitalWrite(22, 0);
        //back motor
        digitalWrite(5, 1);
        digitalWrite(13, 0);

        /////////////// turn left
        digitalWrite(16, 0);
        digitalWrite(21, 1);

        printf("//left front");
    }
    else if (angleDif > acceptErrorAnglePos && angleDif < rightErrorAngle) { //right front
        //front motor
        digitalWrite(17, 1);
        digitalWrite(22, 0);
        //back motor
        digitalWrite(5, 1);
        digitalWrite(13, 0);

        /////////////// turn right
        digitalWrite(16, 1);
        digitalWrite(21, 0);

        printf("right front");
    }
    else if (angleDif < leftErrorAngle) { //left back
        //front motor
        digitalWrite(17, 0);
        digitalWrite(22, 1);
        //back motor
        digitalWrite(5, 0);
        digitalWrite(13, 1);

        ///////////////
        digitalWrite(16, 1);
        digitalWrite(21, 0);

        printf("left back");
    }
    else if (angleDif > rightErrorAngle) { //right back
        //front motor
        digitalWrite(17, 0);

```

```

        digitalWrite(22, 1);
        //back motor
        digitalWrite(5, 0);
        digitalWrite(13, 1);

        ////////////
        digitalWrite(16, 0);
        digitalWrite(21, 1);

        printf("right back");
    }
}

/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*:: Function prototypes ::::::::::::::::::::::::::::::::::::::::::::*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
float deg2rad(float);
float rad2deg(float);

float distance(float lat1, float lon1, float lat2, float lon2, char unit) {
    float theta, dist;
    theta = lon1 - lon2;
    dist = sin(deg2rad(lat1)) * sin(deg2rad(lat2)) + cos(deg2rad(lat1)) * cos(deg2rad(lat2))
* cos(deg2rad(theta));
    dist = acos(dist);
    dist = rad2deg(dist);
    dist = dist * 60 * 1.1515;
    switch(unit) {
        case 'M':
            break;
        case 'K':
            dist = dist * 1.609344 * 1000;
            break;
        case 'N':
            dist = dist * 0.8684;
            break;
    }
    return (dist);
}

/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*:: This function converts decimal degrees to radians ::::::::::::::*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
float deg2rad(float deg) {
    return (deg * pi / 180);
}

/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
/*:: This function converts radians to decimal degrees ::::::::::::::*/
/*::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/
float rad2deg(float rad) {
    return (rad * 180 / pi);
}

```

```

int main() {
    //init code call init .sh file

    printf("start to set up");
    wiringPiSetupGpio();

    pinMode(17, OUTPUT);
    digitalWrite(17, 0);
    pinMode(22, OUTPUT);
    digitalWrite(22, 0);
    pinMode(27, OUTPUT);
    digitalWrite(27, 1); // enable

    pinMode(5, OUTPUT);
    digitalWrite(5, 0);
    pinMode(13, OUTPUT);
    digitalWrite(13, 0);
    pinMode(6, OUTPUT);
    digitalWrite(6, 1); // enable

    pinMode(16, OUTPUT);
    digitalWrite(16, 0);
    pinMode(21, OUTPUT);
    digitalWrite(21, 0);
    pinMode(20, OUTPUT);
    digitalWrite(20, 1); // enable

    // Speed pins enable
    pinMode(23, OUTPUT);
    pinMode(24, OUTPUT);
    // Preset the speed to REGULAR
    digitalWrite(23, 1);
    digitalWrite(24, 1);

    // pre-defined GPS destination coordinates
    float des_latitude = 42.723349;
    float des_longitude = -84.480728;

    // define current loc for testing use (please comment them out when running)
    float current_latitude = 42.733359;
    float current_longitude = -84.580828;

    //gps_init();
    //loc_t data;

    float gpsNewLat = 0;
    float gpsNewLng = 0;

    mpu.initialize(); // Initialize the DMP in MPU9150
    devStatus = mpu.dmpInitialize();
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
        printf("Enabling DMP...\n");
        mpu.setDMPEEnabled(true);

        // enable Arduino interrupt detection

```

```

        //Serial.println(F("Enabling interrupt detection (Arduino external interrupt
0)..."));
        //attachInterrupt(0, dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
        printf("DMP ready!\n");
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        printf("DMP Initialization failed (code %d)\n", devStatus);
    }

while (1){

    //gps_location(&data);

    //float heading = ((atan2(my, mx)) * 180)/Pi;
    //printf("degree: %6hd\n", heading);

    //printf("lat:%lf lon:%lf alt:%lf\n", data.latitude, data.longitude, data.altitude);
    //printf("%lf %lf\n", data.speed, data.course);
    //printf("a/g/m:%6hd %6hd %6hd %6hd %6hd %6hd %6hd %6hd\n", ax, ay, az, gx, gy,
gz, mx, my, mz);
    //printf("m:%6hd %6hd %6hd\n", mx, my, mz);

    //current_latitude = data.latitude;
    //current_longitude = data.longitude;

    float calAngle = atan((des_longitude - current_longitude)/(des_latitude -
current_latitude));
    float curAngle = loop();
    if (curAngle > -180.0 && curAngle < 180.0) {
        float angleDifference = curAngle; //- (90.0 - calAngle);
        float distanceDifference = distance(des_latitude, des_longitude,
current_latitude, current_longitude, 'K');
        printf("degree: %7.2f\n", angleDifference);
        //instructionLoop(angleDifference, distanceDifference);
    }

    //::::::::::::::::::::::::::::::::::::::::::::::::::::*//
    //:: Remove dat file before aquiring new data ::*//
    //::::::::::::::::::::::::::::::::::::::::::::::::::::#//

    if( remove( "dat.txt" ) != 0)
        perror( "Error deleting file");
    else

```

```

        puts( "File successfully deleted" );

//Wrting the data to a text file
ofstream myfile;
myfile.open("dat.txt"); //Opening a file with filename "dat.txt"
myfile << current_longitude << endl; //Exporting the data to the file
myfile << current_latitude << endl;
//myfile << data.altitude << endl;
//myfile << data.speed << endl << curAngle << endl << angleDifference << endl;
myfile.close(); // Closing the file

//loop();

//sleep(1);
}

}

```

UI Code

index.html

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Raspi UI</title>
  <link rel="stylesheet" type="text/css" id="theme" href="css/index.css">
</head>
<body>
  <center>
    <div id="connect">
      <a onclick="GoTo('ON')">Connect to Engine</a><br>
    </div>
    <div id="controls">
  </div>
  </center>
  <script type="text/javascript" src="js/jquery.min.js"></script>
  <script type="text/javascript" src="js/index.js"></script>
  <center>
    <div>
      <div id="streamdata" style="float: none;">
        
      </div>
    </center>
  </body>
</html>
```

remoteRobot.php

```
<?php
$action = $_POST['GO'];
switch ($action)
{
    case "ON":
        shell_exec('sudo /home/pi/remoteRobot/controller/init.sh');
        break;
    case "FORWARD":
        shell_exec('sudo /home/pi/remoteRobot/controller/forward.sh');
        break;
    case "BACKWARD":
        shell_exec('sudo /home/pi/remoteRobot/controller/backward.sh');
        break;
    case "LEFT":
        shell_exec('sudo /home/pi/remoteRobot/controller/left.sh');
        break;
    case "RIGHT":
        shell_exec('sudo /home/pi/remoteRobot/controller/right.sh');
        break;
}
echo json_encode(array('success'=>1,'direction'=>$action));
?>
```

dataPrint.php

```
<!DOCTYPE html>
<html>
```

```

<head>
    <title></title>
</head>
<body>
<div>
    <?php
        echo file_get_contents('dataStored.txt');
    ?>
</div>
</body>
</html>

```

index.css

```

#connect{
    height: 30px;
    margin-top: 0px;
    text-align: center;
    cursor: pointer;
}
#controls{
    margin-left: -60px;
    margin-top: 200px;
}

.up{
    position: absolute;
    width: 60px;
    height: 60px;
    margin-top: -100px;
    cursor: pointer;
}
.down{
    position: absolute;
    width: 60px;
    height: 60px;
    margin-top: 100px;
    cursor: pointer;
}
.left{
    position: absolute;
    width: 60px;
    height: 60px;
    margin-left: -100px;
    cursor: pointer;
}
.right{
    position: absolute;
    width: 60px;
    height: 60px;
    margin-left: 100px;
    cursor: pointer;
}

```

index.js

```
function GoTo(Direction){
    $.ajax({
        type:"POST",
        dataType:"json",
        async:true,
        cache:false,
        url:"controller/remoteRobot.php",
        timeout:25000,
        data: "GO="+Direction,
        success:function(data,textStatus){
            //alert('sucess');
        }
    });
}
```

init.sh

```
echo "27" > /sys/class/gpio/export
echo "17" > /sys/class/gpio/export
echo "22" > /sys/class/gpio/export
echo "6" > /sys/class/gpio/export
echo "5" > /sys/class/gpio/export
echo "13" > /sys/class/gpio/export

#init left/right
echo "16" > /sys/class/gpio/export
echo "21" > /sys/class/gpio/export
echo "20" > /sys/class/gpio/export

echo "out" > /sys/class/gpio/gpio27/direction
echo "out" > /sys/class/gpio/gpio17/direction
echo "out" > /sys/class/gpio/gpio22/direction
echo "out" > /sys/class/gpio/gpio6/direction
echo "out" > /sys/class/gpio/gpio5/direction
echo "out" > /sys/class/gpio/gpio13/direction

echo "out" > /sys/class/gpio/gpio16/direction
echo "out" > /sys/class/gpio/gpio21/direction
echo "out" > /sys/class/gpio/gpio20/direction

sudo raspistill -w 640 -h 480 -q 5 -o /home/pi/stream/pic.jpg -tl 300 -t 9999999 -th 0:0:0 -n
&
sudo LD_LIBRARY_PATH=/home/pi/MJPG-streamer/mjpg-streamer/ /home/pi/MJPG-streamer/mjpg-
streamer/mjpg_streamer -i "/home/pi/MJPG-streamer/mjpg-streamer/input_file.so -f /home/pi/
stream -n pic.jpg" -o "/home/pi/MJPG-streamer/mjpg-streamer/output_http.so -p 8081 -w /home/
pi/MJPG-streamer/mjpg-streamer/www" &

chown -R pi /sys/class/gpio/gpio27/
chown -R pi /sys/class/gpio/gpio17/
chown -R pi /sys/class/gpio/gpio22/
chown -R pi /sys/class/gpio/gpio6/
chown -R pi /sys/class/gpio/gpio5/
chown -R pi /sys/class/gpio/gpio13/

chown -R pi /sys/class/gpio/gpio16/
chown -R pi /sys/class/gpio/gpio21/
```

```
chown -R pi /sys/class/gpio/gpio20/

echo "1" > /sys/class/gpio/gpio27/value #Back motor enable
echo "1" > /sys/class/gpio/gpio17/value #Back motor IN1
echo "1" > /sys/class/gpio/gpio22/value #Back motor IN2
echo "1" > /sys/class/gpio/gpio6/value #Front enable
echo "1" > /sys/class/gpio/gpio5/value #Front IN1
echo "1" > /sys/class/gpio/gpio13/value #Front IN2

echo "1" > /sys/class/gpio/gpio16/value #Front enable
echo "1" > /sys/class/gpio/gpio21/value #Front IN1
echo "1" > /sys/class/gpio/gpio20/value #Front IN2
```