TITLE OF DOCUMENT:

UVS DEVELOPMENT REPORT

Signatures:			
Date:	**Author**	**Dept.:**	**Signature:**
5/18/2017	Antonio Bermudez Nitesh Panchal Sandeep Chahal Shaad Jafari Tabin Gautam	EE ME CSE ME AE	

Summary:

Enclosed is the report of vision system, visual obstacle avoidance with waypoint navigation and path planning components. Using the GNC Simulink file and color vision model we are able to detect the color and avoid the blue color and stop at the green color at the last waypoint during our demonstration. The approach and method we have taken for path planning is discussed here in the report. All the experiments and the incorporation of vison model to the Simulink is described in these report.

Distribution:		
Institution:	**Dept.:**	**Name:**
The University of Texas at Arlington	MAE,EE,IE,CSE	

CONTENTS

NOMENCLATURE

ROS=Robot Operating System
NUC=Next Unit of Computing
NiMH=Nickel Metal Hydride
LiPo=Lithium Polymer
DC=Direct Current
PPM=Pulse Position Modulation
PWM=Pulse Width Modulation
GPS=Global Positioning System
V=Voltage

LIST OF FIGURES

PROJECT OVERVIEW

It is the multidisciplinary project with five group member from different department. We have two week to get the characterization of camera and used the camera information in the vision Simulink model to extract eh pixel information for avoiding and detecting colours. Similarly, the last week was utilize to implement the path planning method to have more autonomous behaviour for our platform. These measurement is used in the sensor fusion algorithm to perform the filtering task. The main task was to use these algorithm to determine the colour and the pixel values and used in our GNC for successfully avoiding and performing given waypoint navigation. The main objective for this report was to present colour detection and avoidance with waypoint navigation and path planning implementation which is presented in the report below.

1 ASSEMBLY OF CAMERA, PIXHAWK, SONAR, NUC AND PHIDGETS

Camera is connected to the NUC to one of the port via USB cable. Camera is used to get the information what it sees with its given range, where vision model extract required data for performing the colour detection task. Pixhawk was integrated to the system that was designed for Waypoint navigation. The Pixhawk gets power from LiPo battery connected by 3DR Module which gets power from battery and is distributed one to the Pixhawk and other to DC to DC converter for NUC. Toggle switch is used to turn on the system and the Pixhawk for safety reason. The two Phidgets motors gets power from 7.2 NiMH battery and the other end is connected to the left and right encoder motors with 5 pin where yellow and white wired pin provide the signal. There are three pin, ground (Pin 7), 5V power (Pin 6) and analog out (Pin 3) pins of MB1240 sensor which is connected to analog port 0 of the Phidgets. The sensor line is calibrated to match the sensor beam patterns and provides a long range detection zone. The Phidgets motor gets the information from the ROS node which subscribes the message from the GNC block created in the Simulink Matlab file via USB cable. Both Phidgets boards is powered from the same battery with a Y-cable and is other connection cable is screwed into the power ports. Encoder pins for left and right motor are connected in opposite manner (yellow and green), since if connected in same manner would result in one turning in positive while other turns in negative direction with respect to their turn direction.
Intel NUC has both the Linux and Matlab installed in it which can be operated in another PC by team viewer for Windows OS and by hotspot connection for Ubuntu 14.04 Thrusty Linux distribution. In these way we are able to run Simulink /Matlab on the platform during our waypoint navigation. It helps us to run the program and debug it when performing the evaluation of the platform easily and faster.Matlab supports communication with the motors and s different sensor connected to the NUC either through direct connection or through ROS .This way the GNC algorithms can be run with actual sensor data while the vehicle performs actual mission by the actuator commands produced by the program/code written.

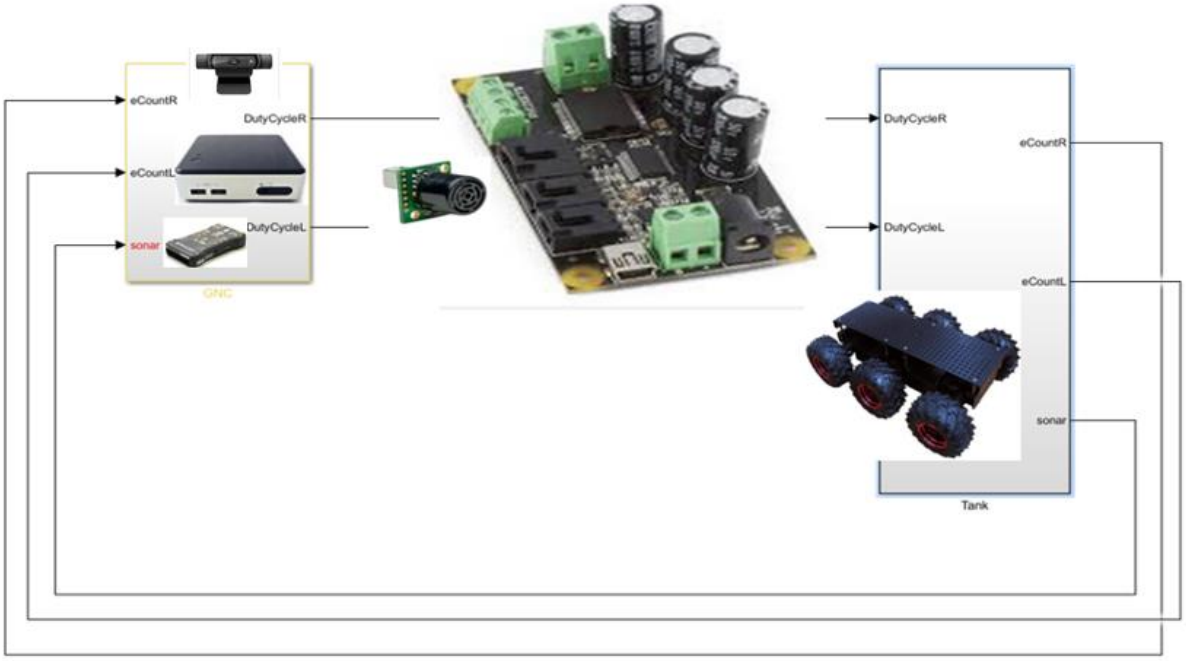

Figure 1: Integration of Camera, Pixhawk, Sonar, NUC and Phidgets

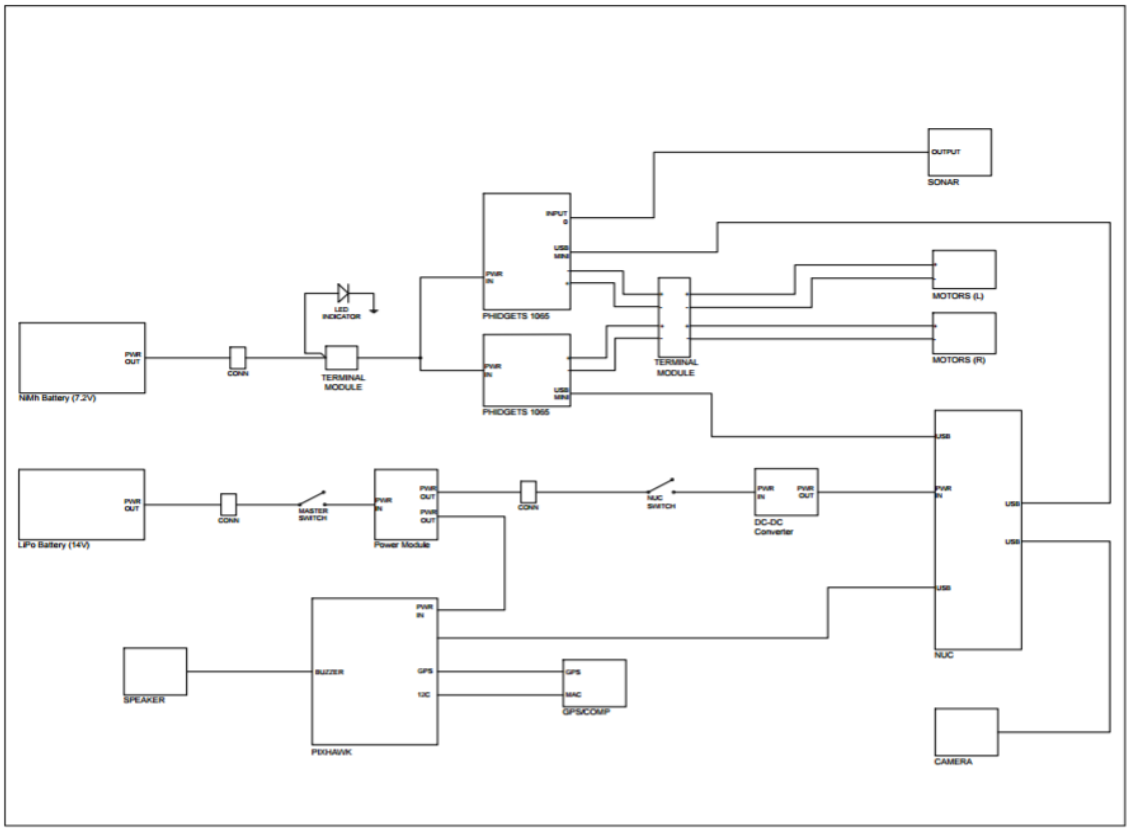

Figure 2: Circuit Diagram

Figure 3: Physical layout of Vehicle

2 CAD MODEL

Figure 4: 3D View of Rover Assembly

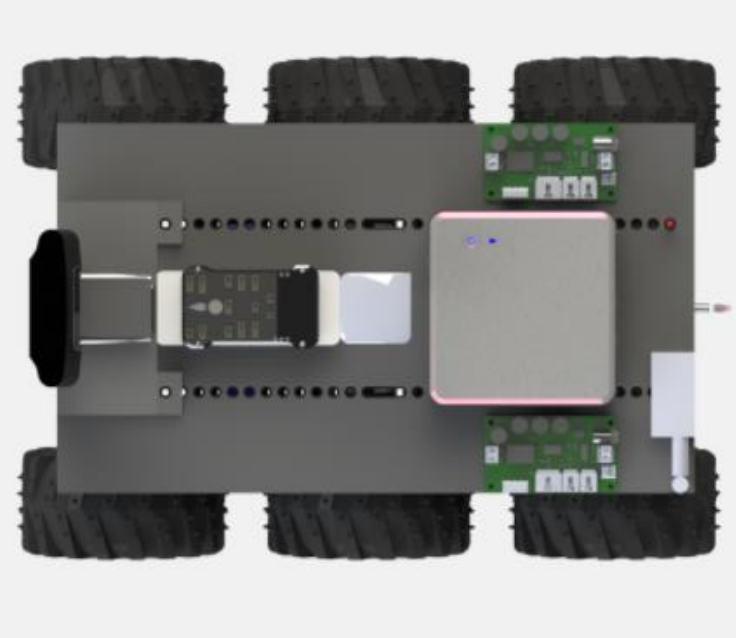

Figure 5: Top view of 3D Rover Assembly

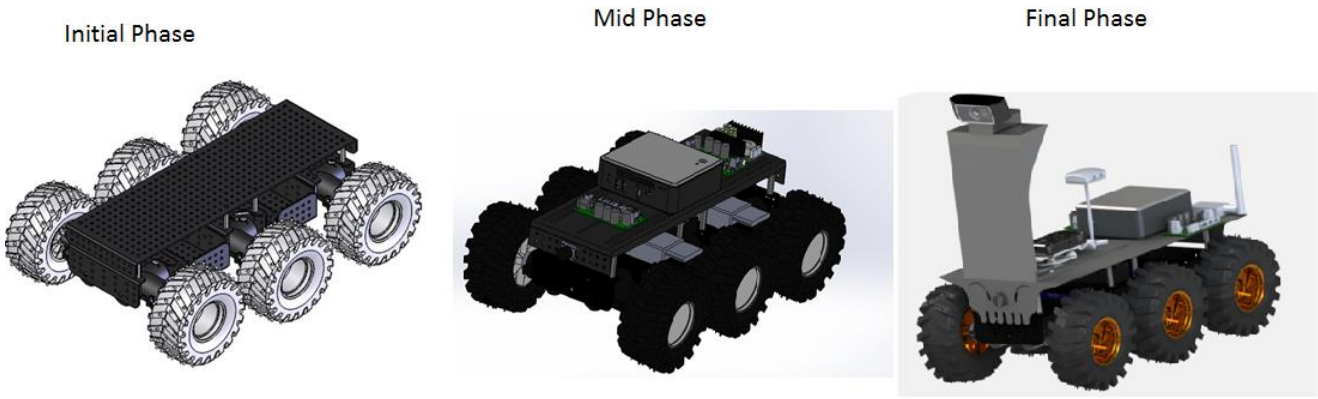

Figure 6: Transition of CAD Model

3 BATTERY

NiMH of 7.2 V is used to power six motors and two Phidgets motor boards. Lithium polymer (LiPo) ion of 14.1 v is used to power NUC via DC to DC converter which get 19V to operate. Lithium batteries are the preferred power sources for most electric modelers today. They are similar to Lithium Ion

batteries in that they each have a nominal voltage of 3.6 volts. They offer high discharge rates and a high energy storage/weight ratio. LiPo battery should be handled very carefully while charging.

Figure 7: NiMH/LiPo battery

4 NUC

NUC designed by Intel with Intel® Core™ i5-4250U processor basically,is small computer with central processing unit .It has got 10 GB Samsung SSD with 16 GB DDR3 with the feature of intel WIFI/ Bluetooth card in it .The resistor is used once for connecting into one our personal computer. The NUC is connected to both the motor controller via USB. The Matlab Simulink is inside the NUC is connected to the Phidgets motor via ROS node connection by publishing and subscribing. The NUC gets power supply from 14.8v LiPo battery which converts it to 19 v from DC to DC converter.

5 DC TO DC CONVERTER

It use to convert the voltage from the LiPo battery which is 14.8v to 19V.The requirement for the NUC power supply is 19v. The power and ground supply is connected to input side of the Converter where as the output is connceted to the power cord.Safey precaution must always followed to insure that NUC is not receiving more than 19v. Every naked wired connection must be insulated, heat sink is the best possible way to insulate them in lab.

6 PHIDGETS MOTOR CONTROLLER

Two Phidgets motor is used to control the PWM provided to the middle motor both on left and right wheel respectively. The Phidgets motor has the ability to control the direction, velocity and acceleration of motor. The motor is powered by 7.2 v NiMH and is connected to board by Y cable, current flows through the motor, and it will begin rotating. Depending on the direction of the current, the motor will rotate clockwise or counterclockwise. Switching the voltage very quick the controller is made smaller, more efficient, and cheaper. It is connected to the NUC through USB and it enables MATLAB to drive motor, and enables it to read an encoder counts up to two analog and two digital sensors.

Figure 8: Phidgets motor Controller

7 SONAR RANGE SENSOR

The MB1240 sonar range sensor is used to detect the obstacle ahead of it and send the range information to the motor controller to change the direction and speed. There are three pin, ground (Pin 7), 5V (Pin 6) and analog out (Pin 3) pins of MB1240 sensor which is connected to analog port 0 of the Phidgets. The sensor line is calibrated to match the sensor beam patterns and provides a long range detection zone. The sensor has the highest noise tolerance and calibrated to provide reliable range information to large targets even in environments with strong acoustic or electrical noise sources.

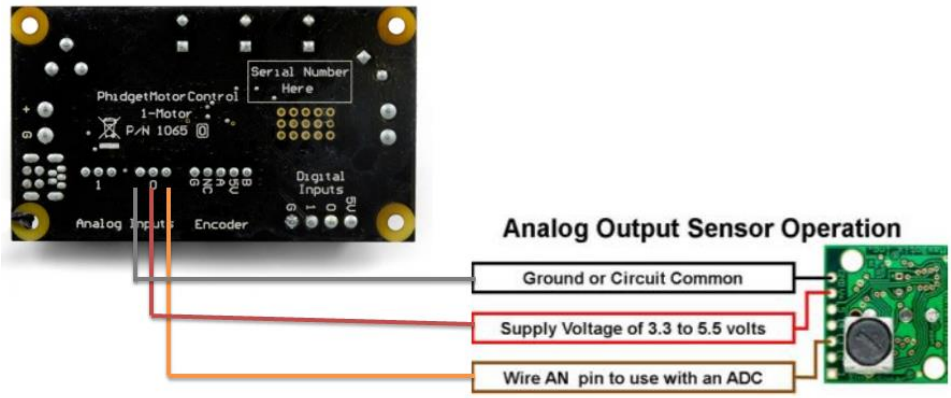

Figure 9: Connection of sensor with the Phidgets

8 PIXHAWK

Pixhawk is used to get the sensor information which is used by Simulink model for sensor fusion. IMU, GPS and accelerometer sensor are fused in Pixhawk for getting the orientation, velocity and acceleration of the platform Pixhawk is the open hardware that is available to everyone like student or industry who are working in autonomous system. It provides hardware that is capable of autopilot and run efficiently with the real time operating system. It can be used any system like copter, drone or in a rover where we are working on it now. Buzzer and safety switch is connected inorder to arm and disarm pixhawk manually when operating rover. It is conncected to pixhawk connection port with the 4 pin and two pin for switch and buzzer respectively.GPS and Compass consist of 3DR GPS along with

compass to provide autopilot with the positioning data of rover. It is conneced by 6-wire cable to GPS ports and to the I2 C port using the 4-wire cable.

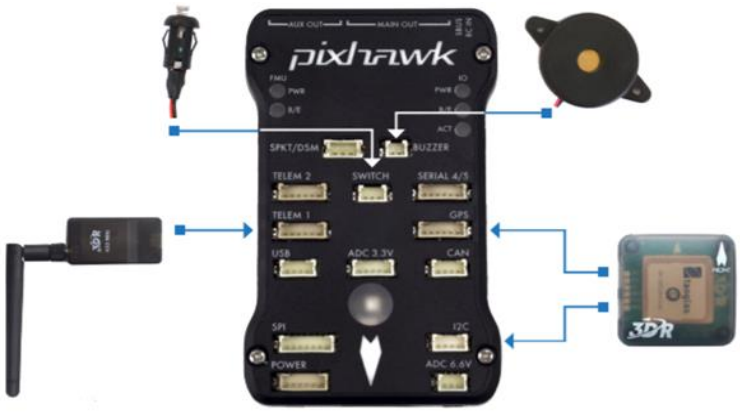

Figure 10: Pixhawk with major components

8.1 GPS AND COMPASS

It consist of 3DR GPS along with compass to provide autopilot with the positioning data of rover. It is conneced by 6-wire cable to GPS ports and to the I2 C port using the 4-wire cable. The Pixhawk consist of IMU sensor where we angular orientation, acceleration and velocity is obtained. Pixhawk utilize its internal GPS as a primary compass and use the external compass connected to it as a secondary compass in case of loss to the connection.

9 ENCODER MOTORS

This rover consists of two dc brushed motor with encoders and 4 dc brushed motor without encoders. This gear motor consists of a high-power, 6 V brushed DC motor combined with a 34:1 metal spur gearbox, and it has an integrated 48 (Cycle per Revolution) CPR quadrature encoder on the motor shaft, which provides counts per revolution of the gearbox's output shaft. The gear motor is cylindrical, with a diameter of 25 mm and 4mm of output shaft, whereas the motor without encoder are used to provide higher speed are installed on front and rear part of the rover.

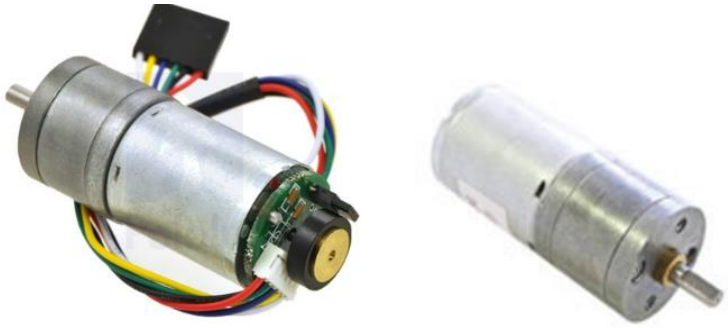

Figure 11: Motor & Motor with Encoder

10 VISION SENSOR

Logitech C922x Pro Stream Webcam Camera is used as vision sensor to get the information from the colour that is in the range of the camera view. It is connected to one of the port available at the NUC by USB cable.The camera has the mixing applications with automatic low-light correction and autofocus two built-in omnidirectional mics with noise reduction for clear stereo sound and for smooth uploads using less network bandwidth.

Figure 12: Camera

11 TOGGLE SWITCH

We have incorporated the toggle switch for safe and easy connection and disconnection of power from battery to system. Toggle switch is connected in series with the 3DR power module which provides power to motor controller and to Pixhawk.

Figure 13: Toggle Switch

12 VISION SYSTEM

Vision system is incorporated by using the vison model provided by professor in the Matlab/Simulink. To access the webcam from Simulink we installed called Image Acquisition Toolbox supported for generic video interface which is a feature Matlab has provided as an additional support package. First task was to extract different color by changing the HSV constant value in the median filter.

Since the camera is mounted at certain height from the platform base at certain angle the location in the image of points touching the ground indicates how far the object is from the vehicle. The image obtain in Simulink in video viewer indicates that image moves up with the increase in distance. Therefore good calibration need to be done between the image obtain and the position of the camera. So we need to determine the calibration between the image seen and physical presence of the color object which is basically done by using the pixel obtain from vison model and distance at which that pixel is present. After calibrating these image coordinate and location in the real world we can use the algorithm to avoid the color and stop any given color .Camera is placed at the height of 17cm and fixed at angle of 22.5 degree so that it can cover its range of view suited for avoiding the color.

So to determine the location of the image the vison system first filters the objects by removing noise, normalizing and emphasizing the image, second extraction of feature like edge, color and texture, thirds segments into a region that belong to a common objects, and finally identify the object i.e. recognition. In our case we have use the extraction of color for identifying the object making our system simple for obstacle avoidance.

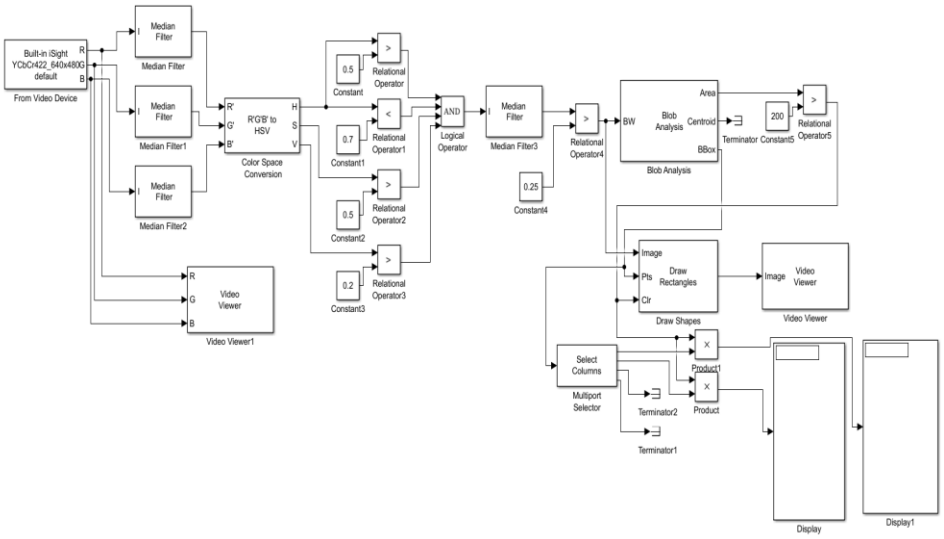

Figure 14: Vision Model

After filtering the image capture by camera and converted it into HSC value the color is extracted and the median filter is used to contain the value take the median value of neighboring pixel. Blob analysis is carried out to extract the feature like centroid, area and bounding box. The blob analysis block is used to calculate pixel value for labeled regions in a binary image. Area represent the number of pixel in that region, centroid gives m by 2 matrix of centroid of coordinate where m represent the number of blob, Bounding box provides m by 4 matrix of bounding box coordinates. The block provides output quantities such as the centroid, bounding box, label matrix, and blob count. Using blob count, image

from output filter rectangle is drawn and viewed in video viewer we get output as pixel value from the top of the image and from the left of the image.

12.1 CAMERA CHARACTERIZATION

First thing we have to decide was the positon and the angle of camera to have the desired performance of avoiding obstacle. We initially went for height of 17 cm and angle of 36 degree with the resolution of 320 by 240 pixel the processing time taken by the system was higher .The maximum distance camera see is 81.28 cm and minimum distance is 20.32 cm, any object will below 20.32 cm will be in blind side. So we have decided to reduce the resolution of 160 by 90 pixel and ultimately with change in camera angle of 21.5 degree for better view, and the processing time was faster than with the earlier configuration. The field of vision is increased so that camera can now see up to 208.28cm with just small compromise in the shortest distance it can see which is 25.4 cm. Below is the result for the mapping of pixel for each resolution and configuration. If we observe the curve fit for both resolution they approximately follow second order polynomial equation. In order to make code simple and faster we have assumed linear relation of distance and pixel in algorithm. After integrating these algorithm we did not find significant effect by making the linear and we were able achieve acceptable result.

160 by 90		320 by 240	
pixel	distance(inch)	pixel	distance(inch)
87	10	235	10
77	12	203	12.25
72	12.8	161	14.75
57	15.5	130	17.5
47	18	100	20.25
40	20.5	87	23
32	24.5	25	36
24	28	10	40.25
18	32	4	40.5
12	35.75	137	47
9	38.75	143	49
7	41.25	1	42
5	43		
3	45		
1	47.25		

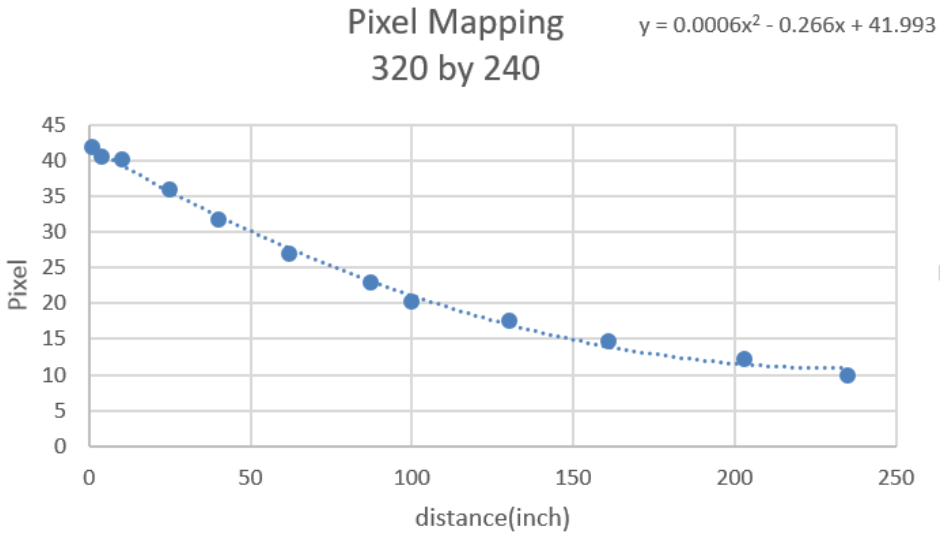

Figure 15: Pixel Mapping 320 by 240

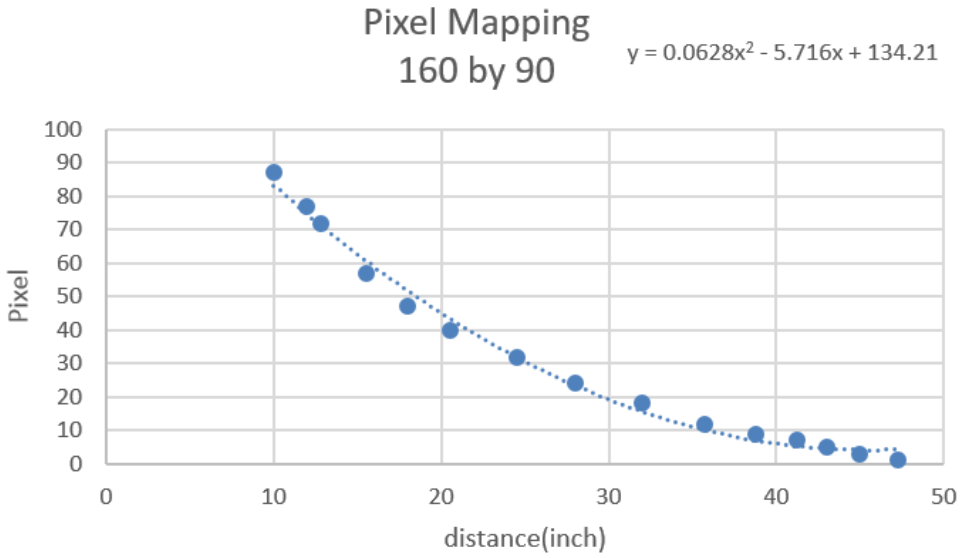

Figure 16: Pixel Mapping 160 by 90

12.2 COLOUR EXTRACTION

Color based extraction is done by using the common method that is done in machine vision. First the filtering is done by reducing the noise in individual pixel i.e. blurring the image. Second is feature detection by identifying the pixel that are within a given color range and converting into map of black and white grayscale model. Finally, segmenting is done by linking the neighbor pixel that are within the same range by bob analysis.

HSV color system using the three components of Hue, Saturation and Brightness is the closest to human eyes and is most suitable for image processing. Converts the image into a grayscale image by taking the average of the R, G and B level and using that as the grayscale pixel level. This will create

a grayscale image. Color to gray processing converts a color image into a grayscale image by using a chosen range of colors as the maximum brightness. The rest of the colors get converted to a range of grayscale values based on the selected colors after using the median filters.

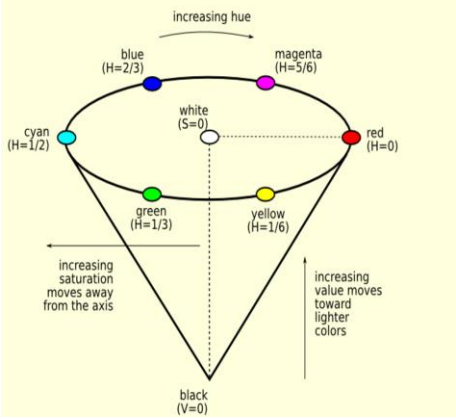

Figure 17: HSV Scale

Based on the HSV scale above the value for regional operator is determined for each color after RGB value is converted to HSV value. During our experiment we found Hue (H) representing the true color to be from 0.9 to 1 and Saturation(S) to be greater than 0.5, Brightness (V) to be greater than 0.1.Similarly the value for other color is tabulated below. Since our obstacle avoidance color is light blue so any saturation value greater than 0.1 instead of 0.5 will be able to extract light blue color.

Color	H	S	V
Red	0.9-1	>0.5	>0.1
Green	.55-.6	>0.5	>0.1
Yellow	0.1-0.2	>0.5	>0.1
Blue	.35-.45	>0.1	>0.1

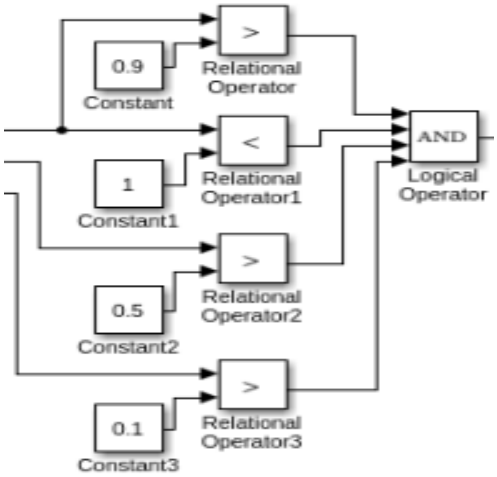

Figure 18: HSV for Red

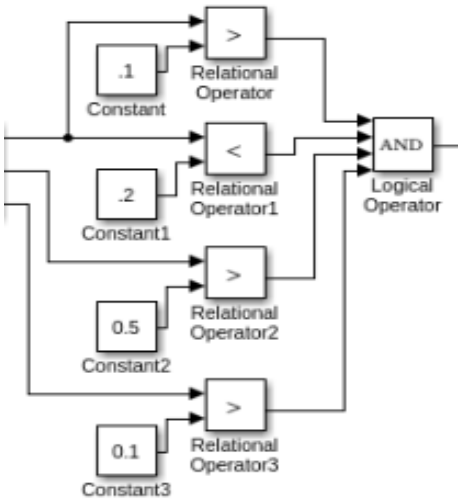

Figure 19: HSV for Yellow

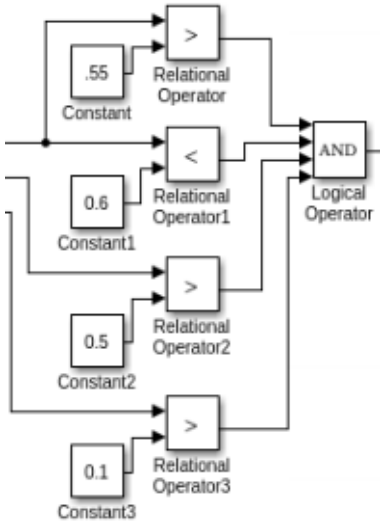

Figure 20: HSV for blue

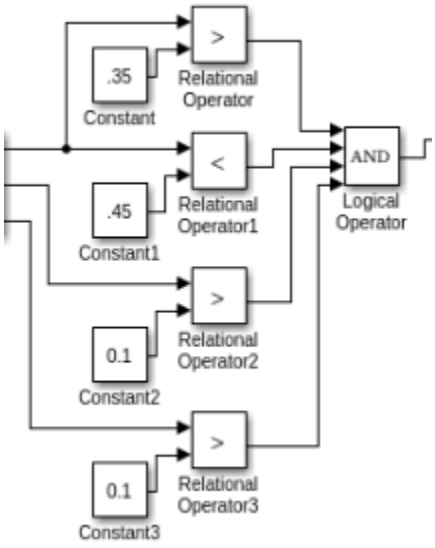

Figure 21: HSV for Green

After getting the object extracted as color blob we utilized the output like centroid, area and bounding box to see the variation in pixel and compute the location and distance of the object from the platform. We know that lowest pixels value are from the ground, we utilized the pixel value of color object from top and from left and using the camera calibration location and translated to the floor location

13 VISUAL OBSTACLE AVOIDANCE WITH NAVIGATION

Since our sensor data consisted of images, we used image processing to accomplish our goal. Our algorithm was based on image segmentation, augmented with median filters to eliminate noise and improve the segmentation algorithm accuracy. If there was a color in front of it, it would navigate to avoid the color; if it sees the target color it will try to move towards it or stop if close enough; otherwise, it will simply go straight forward. Our platform equipped with a webcam, Sonar sensor, with GNC setup allowing it to go through the given waypoint through dead reckoning system. The goal of this project is to avoid any particular color obstacles while navigating a waypoint in an indoor environment i.e. in front of Nedderman atrium and stopping at particular color at last waypoint. For this project, we used only webcam and Simulink model median filters to smooth the image and then applied image segmentation on the smoothed image; we then decided which of the segments was the floor and used the position of other segments to see which the colors were and avoid them. We used a simple color filter to find the object and programmed our rover to move sideways to avoid color. We tested our code with a variety of color and we were successful to avoid the color and stop at any particular color.

At beginning of the visual obstacle avoidance we tried to work with open CV but was not able to get the desired result since the output where in multiple array and couldn't understand the result to utilize and moved with direct integration with the Matlab. Similar concept was utilized as we did for sonar sensor obstacle avoidance i.e. if it sees the color it at certain distance (pixel) it will make change of 2 radian left or right depending on where the location of pixel is located. Below is the step that we followed for the integration of vision system with navigation.

13.1 SUBSYSTEM

First step we did is we created a sub model that will input from RGB color space conversion and converts into HSV value. The main output from these module is to extract blue and green color for avoidance and goal respectively.

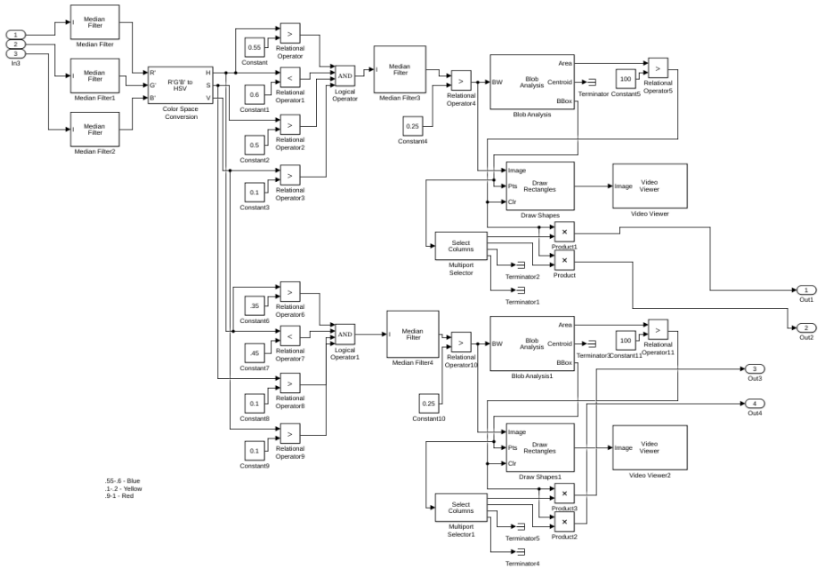

Figure 22: Inside of Subsystem

Since vision model gives two output display one for the pixel value from the top and other from the left of the image , so we have four output with two for the pixel value from top and two for the pixel value from left for both the blue and green respectively. Our vision model only extract one color at a time depending of the HSV value the output is displayed either for the blue or green color. If it detects blue color displayed value will be only from output 1 and 2 where output from 3 and 4 is zero and vice versa when it detects green.

13.2 OBSTACLE AVOIDANCE AND GOAL

Since our obstacle being blue we created a Matlab function where it takes input from subsystem output i.e. display and display1 with output of y. Once the colour is detected we made a logic that if it sees the blue colour make a turn left or right depending upon the pixel value else move in a straight line by setting output of y=0. If the camera see the left side of centre that pixel value greater than 100 than it will make left turn and if it see right side less of pixel value less than 80 it will make right turn. Matlab function for green colour is shown below.

```
function y = fcn(display1,display)
if (display1<=80) && (display>=15)
   y=4;
elseif (display1>100) && (display>=15)
   y=2;
else
   y=0;
end
```

Similarly when the camera vision system extract the color green from vision model once it pixel value lower than at pixel located at center of goal i.e. with the pixel value of greater than 80 it will stop the simulation by using the stop simulation block.

```
function y = fcn(display1,display)
if (display1<=50) && (display>=2)
   y=2;
elseif (display1>100) && (display>=2)
   y=4;
else
   y=0;
end
```

13.3 COLOUR MATLAB FUNCTION

An output from each colour is connected to Matlab function which gives single output either blue or green colour. A colour Matlab function is created to give used as input for obstacle function .Depending upon the output from each colour the system provides output value of that colour Matlab function. Simple logic is created if the blue value is greater than zero output will be blue and same logic is applied for green.

```
function y = fcn(blue,green)
if blue>0
   y=blue;
elseif green>0
   y=green;
else
    y=0;
end
```

Output from the camera or colour Matlab function is utilize in obstacle Matlab function to have output of either camera or sonar sensor depending upon which sensor gets activated. Similar logic is applied in these function as to the colour Matlab function. If the output from the camera is greater than zero than it will have output of camera and same applies to sonar.

```
function y = fcn(camera,sonar)
if camera>0
   y=camera;
else
    y=sonar;
end
```

13.4 NAVIGATION SUBSYSTEM (FINAL DECISION)

The navigation subsystem consist of all the Matlab function discussed above which will give output of either sonar or camera depending upon which gets activated. If the sonar sees the obstacle at certain distance it will make a turn avoiding obstacles and if camera sees the colour it avoids if it's blue and stop if it is green. The change in angle is subtracted from the computed headed angle to make the required change in angle to make avoidance in the navigation block as shown below. The output 2 is connected to the change in heading angle.

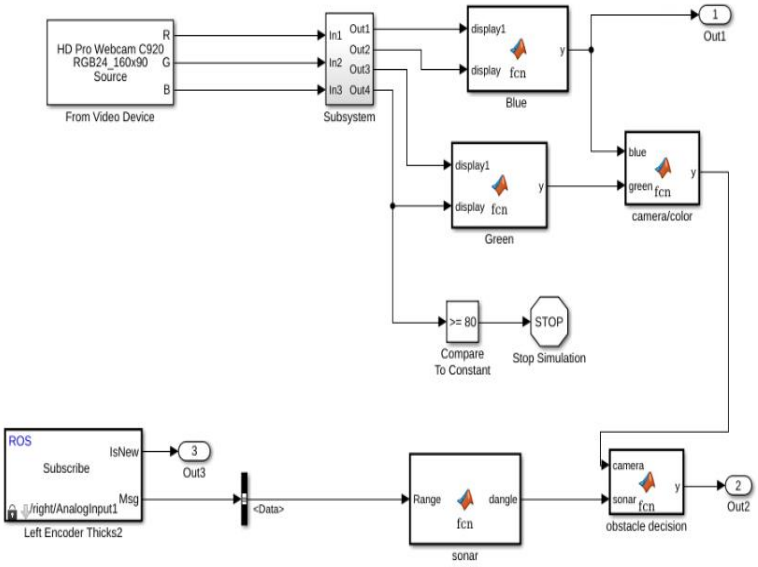

Figure 23: Navigation Subsystem (final Decision)

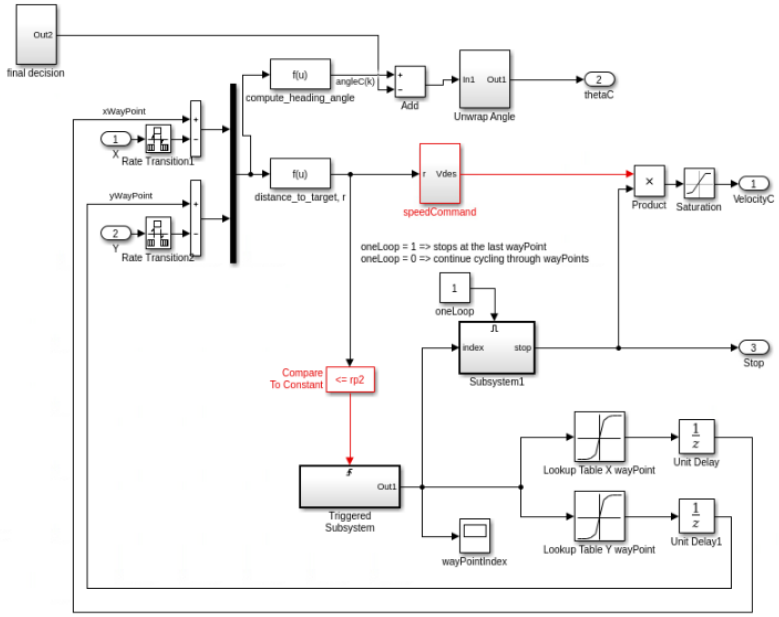

Figure 24: Integration of Subsystem inside Navigation

The navigation subsystem output is connected to computing heading angle for desired turn to avoid the obstacle. So the changes made inside the navigation is done only in changing the angle which is achieved by taking the output from each sensor depending upon which gets activated.

13.5 EXPERIMENTS

We tested our platform on three courses: a simple straight line avoiding blue color, another straight line avoiding the blue color and obstacle in front of it and some waypoint where it need to make turn and need to perform the color and obstacle avoidance. These experiment was carried out inside the lab and other in the hallway. We ran many trials on each course; without targets, color avoidance was almost perfect , the only time it did nothing was when the when the vehicle was not moving to the direction where color was placed i.e. it was not seen at all by the camera, this only happened once or twice. Overall, the success rate of Platform being able to avoid color and stop at particular color was great. We also did a few tests in very different environments: specifically, in a hall with a reflective floor outside of Nedderman Hall room 250. We found that the code behaved very poorly on a reflective floor: reflections caused the variation in the pixel value so program would try to go through color since it was made to avoid obstacle at particular range of pixel. We found lag in the response time of vehicle since the color detection was incorporated directly into the GNC Simulink we made modification like changing the resolution and increasing the response time and also made changes in the pixel value so that it will be able to avoid the color. Some simulation result is shown below where platform avoids two blue color and reaches to way points, only three way point is shown to demonstrate the result.

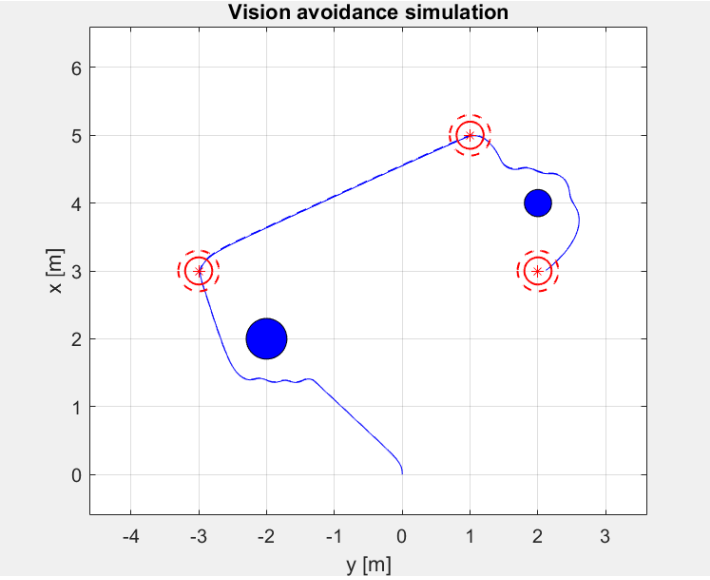

Figure 25: Visual avoidance simulation

14 PATH PLANNING COMPONENTS

Path-planning requires a map of the environment and the robot to be aware of its location with respect to the map. The path planning stage involves the search for a collision free path, taking into consideration the geometry of the vehicle and it surroundings, the vehicle's kinematic constraints and any other external constraints that may affect the planning of a path. Path planning for our vehicle is divided into two main categories static and dynamic path planning. Static path planning require an obstacle map of the world (vehicle environment).The path is pre-computed and then given to the vehicle to execute the path. The most common map is the occupancy grid map where, the environment is discretized into squares of arbitrary resolution .Although it's easy to use but the grid maps requires large memory and computational time to traverse data structures with large numbers of vertices .External sensor like camera are very integral part of guidance and navigation for determining the presence of an obstacle, location of obstacle and object in the environment (environmental mapping) and determining where the vehicle is in the world (localization).We have already used sonar sensors for obstacle avoidance in previous module of obstacle avoidance and waypoint navigation which are used for determining the presence of an obstacle and distance to the obstacle. But they does not provide the information like differentiating multiple object and identification of object. We can use the sonar sensor for mapping the world but the information is limited by the size of sensor cone. Since, the data from these sonar sensor is not enough to have good map, so to enhance the mapping camera sensor is used i.e. logic tech web camera is used in our platform for better information of world mapping.

Two new component replaces the navigation waypoint system one is the waypoint generator another is path planning. We have implemented Manhattan distance navigation function as path planner since its easy and simple to construct. Since in previous week we have use the camera for avoidance we will use camera and sonar to avoid the obstacle that is ahead of it so that system have world map with obstacle and waypoint. The new system will have the vision avoidance model that we used in last week is incorporated.

14.1 CAMERA CALIBRATION FOR MAPPING

Since the mapping of the height (Y) is already done for previous the only remaining thing was mapping of X (width) .The maximum distance it can see with the current setting is 47.25 inch i.e. pixel value of 1, we measured the width the camera covers in real world. The distance from pixel value of 1 to 143 and is used these information to obtain correlation .The relation is to use to obtain the distance from 1 to 160 and use to map into the grid by using the heading angle and state of the vehicle just before it sees the obstacle to obtain the path plan.

Width for 160 by 90 Resolution	
pixel	distance(inch)
1	0.0001
3	3.5
19	8.75
47	17
61	21
78	26.5
87	30
104	35
124	42
137	47
143	49

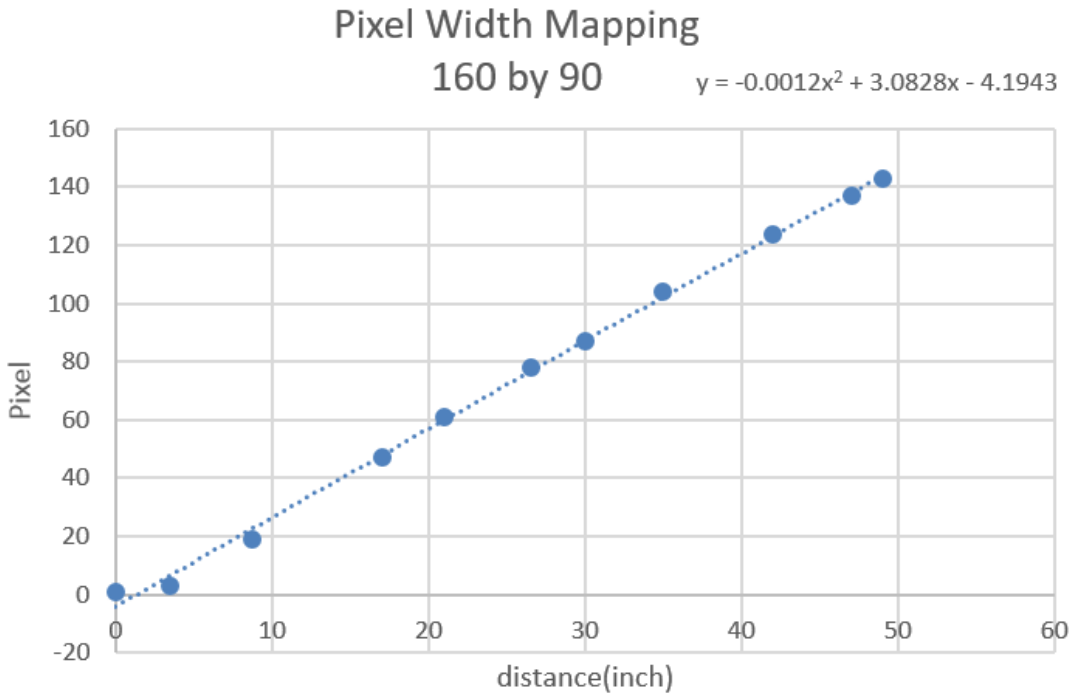

Figure 26 Pixel Width Mapping

14.2 DYNAMIC OBSTACLE MAPPING

The camera sees the world as shown in picture above and gives out an image. This picture is converted in matrix so that we can play around with it. The camera gives an image in RGB format, this image needs to be converted in HSV to detect colors. Simulink median filters can be used to do these task. We measured the distance of the frame the camera is capturing, then we took the real world distance and use these information to map the dynamic obstacles .Pixel mapping is utilized, when camera see an obstacle its height and width cover in real world is mapped into the occupancy grid where we will locate the position of the obstacles.

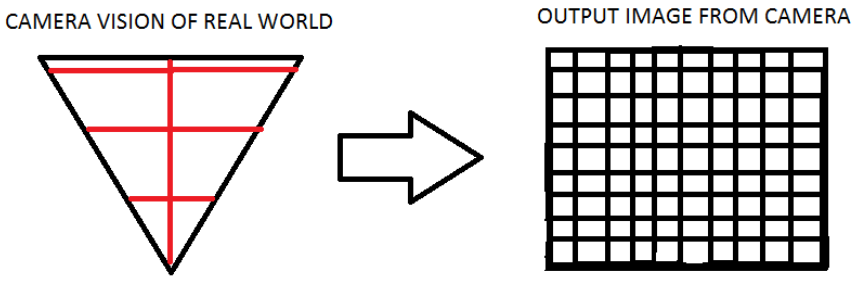

Figure 27: Camera Vision vs Camera Output

The main problem with it was the pixel near to the camera are covering less distance and top pixel are covering more distance. To overcome this problem we detected the obstacle in top row as shown in camera vision of the real world and converted that into pixels. So we took three measurement as shown in red in above picture and we measured the distance for each pixel value. Now, we know how much each pixel is covering that distance in real world and use in our world map created in our Simulink model. We now have the height and width of the obstacle and utilizing these values, and is converted that into our grid location. The flag world changed was set to 1 and it would call obstacle function, which would map the obstacle into the grid, with static obstacles and use the old grid to update the map by placing the obstacles seen by camera into grid .We rerun the path planner which would give us the path to follow.

14.3 SIMULINK MODEL

New Simulink model with waypoint generator and path planning is added to the existing guidance model in the sonar we have before with the lookup table of x and y waypoints. Everything is in one Simulink model with the ROS Topic already presented for different GNC Components. The only change to previous Waypoint navigation is in the guidance component. Way Point generates the random waypoint and best waypoint is chosen and given to the path planner where it will construct the roadmap towards goal. Simple waypoint mechanism has been replaced with a path planner and a waypoint generator.

Data store memory element is created so that it can store the information generated like Occupancy grid, Path length, Cell size, Grid origin and interact and exchange these data between Simulink and Matlab code. Cell size provides the information about how large each cell will be in the occupancy grid and Grid Origin defines location of the outside corner of grid cell origin point. Once the world changed flag is raised the path planner gets trigger and using the known occupancy grid of obstacle location and the location of goal a path is generated and stored in pathlen. Once it receives the value in pathlen new path flag is raid indicates its previous value has been changed where, waypoint generator takes the path and set it as first waypoint. After setting the first point as waypoint and the when it reaches to the maximum distance defined in waypoint generator it will indicate to go to next waypoint until it reach the goal which will be the last waypoint. Since there was already a waypoint generator code we just kept it as it and implemented a navigation function for path planning inside the path planner block, we experimented to use Manhattan navigation function to develop a path that reaches to goal location. The navigation block is connected to previous GNC where x-position and y position along with theta being connected from guidance block to navigation block that contains the waypoint generator and path planner. In order to start the Rostopic start message we need to give any message to ROS to start the Simulink block.

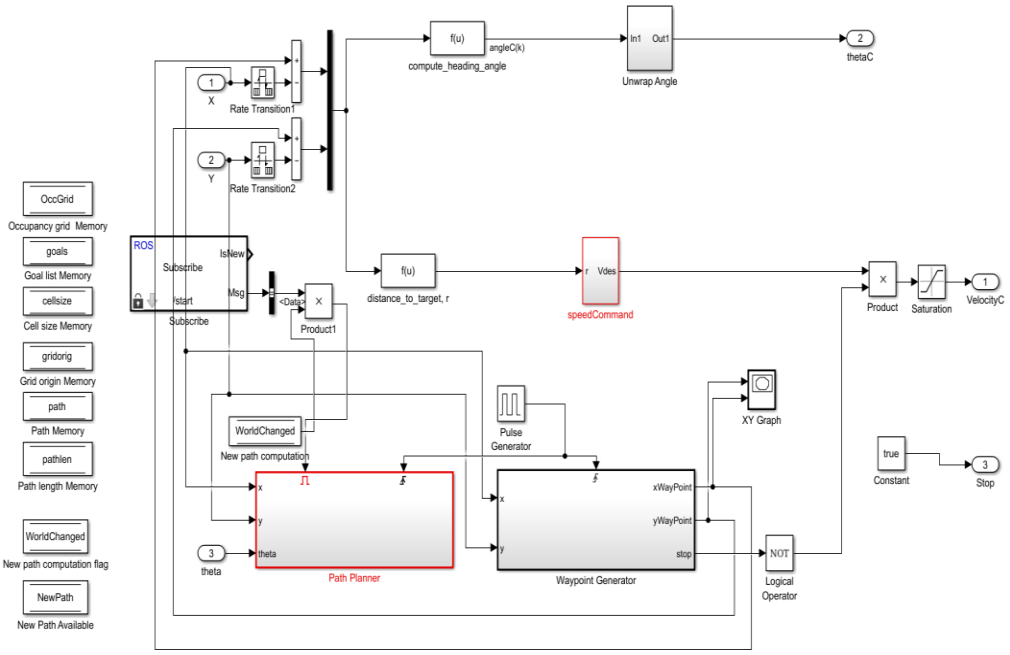

Figure 28: Waypoint generator and path planner

14.4 MANHATTAN DISTANCE

Having the graph of the map, all that remains is to choose and implement some suitable algorithm. There are many algorithms and their variations able to find the shortest trajectory but they are can differ a lot. Even though they would eventually find the same result some are much faster because they expand less nodes or require less mathematical operations. We decided to go with the Manhattan distance method for calculating the short distance to the goal since it easy to code and implement into our model. Path-planning requires a map of the environment. Vehicle and obstacle will have its location with respect to the map where grid maps require large memory where it can store the location of the vehicle, obstacle and goal .The map gets updated every time vehicle see any color obstacles and takes large computational time to traverse data structures with large number of vertices. Path planning is done by taking the distance between two points in a grid based on a strictly horizontal and or vertical path and is opposed to moving in a diagonal way. Manhattan distance is a very simple method where space is divided into grid where cell contains goal, obstacle and location of our platform. Free cells are used to construct path avoiding obstacle. Therefore obstacle cells are made larger so that when vehicle makes turn it can avoid the obstacle in real world. Starting from goal the distance increases using four connected neighborhood and increase till it reaches its current location. Shortest distance is taken and back traces the path until vehicle reaches goal.

14.5 PATH PLANNER (MANHATTAN DISTANCE)

We have use Manhattan distance for path planning. A world map of 30 by 30 is created with each cell equal to 0.5 m. The goal location is updated in the world map(x, y) and set the value of equal to

3.Whereas, for every obstacle world map is set to 1 and boundary of obstacle is set to value of 2 and remaining is set to zero(free space).Since occupancy grids are some of the easiest ways to model the obstacles ,occupancy grid is created which is easy to use , store and integrate with path planning .In our algorithm for Manhattan distances we have made occupancy grid equal to world map with the initial grid location of (0,0).After that Manhattan grid is created that of size off occupancy grid where obstacle are represented in set of grid cells by having very high value i.e. infinite. Here infinite represent the 1 and 2 of occupancy grid and we set the goal location with zero. Using four connected neighbourhood the adjacent values of cell are increase by 1, taking care of the boundary condition i.e. only value inside the grid is increased. It keeps on increasing until all the cells are occupied with number. Using the state of the vehicle i.e. location of vehicle obtained from odometer, approximation of the overall space into world map path is selected. For example if we starts our vehicle at the (-0.5, -0.5) it chooses minimum distance to reach goals. It stores the path travelled by the vehicle to reach the goal (i.e. shortest distance/ path).The Manhattan distance Matlab program integrated in path planning is shown below.

```
%function prepared_world_map= constructing_worldmap_obstacle()
world_map=zeros(10,10);

%goal
world_map(8, 7)=3;

%obstacle one
world_map(size(world_map,1)-2 : size(world_map,1), 3:4)=1;
world_map(size(world_map,1) - 3, 3:4) = 2;
world_map(size(world_map,1) - 3:size(world_map,1), 2) = 2;
world_map(size(world_map,1) - 3:size(world_map,1), 5) = 2;

%Obstacle 2
world_map(2:3, 7:8) = 1; % obstacle
world_map(1, 7:8) = 2;  % extended obstacle
world_map(4, 7:8) = 2;  % extended obstacle
world_map(1:4, 6) = 2;  % extended obstacle
occuGrid=world_map;
%fucntion
ix=1;    %initial x location in the grid
iy=10;    %initial x location in the grid
heading=0;
%k=find(world_map==3);
[row,col] = find(world_map==3);
gx=col;
gy=row;
```

```
manhatten_grid = zeros(size(occuGrid, 1), size(occuGrid, 2));
manhatten_grid(occuGrid==1)=Inf; % setting manhatten grif to infinite at places og obst
manhatten_grid(occuGrid==2)=Inf;
parse_next=0;
%logical(sum(manhatten_grid == 0)~= 1)
a=numel(manhatten_grid);
while (a-nnz(manhatten_grid))~=1
    if parse_next ==0

        % function manhatten_grid = mark_cells(y,x,parse_next,manhatten_grid,occuGrid);
        distance = parse_next;
        if (row ~= size(manhatten_grid, 1))
            if (manhatten_grid(row + 1, col) == 0) && (occuGrid(row + 1, col) == 0)
                manhatten_grid(row + 1, col) = distance + 1;
            end
disp('Way points')
%disp(path)
path_matrix=zeros(size(manhatten_grid));
for node=1:length(path)
    temp=path(node);
    a=temp{1}(1);
    b=temp{1}(2);
    disp([a,b])
    path_matrix(a,b)=1;
end
disp('PATH')
disp(path_matrix)
```

14.6 SIMULATION RESULTS

The first step in path planning is choosing a map representation that is appropriate to the application. Since the professor has already defined the location of the goal and main obstacle, we have used the algorithm accordingly. The second step is finding the method and algorithm for shortest path. Using these algorithm we have carried out the simulation where the results is presented below. After making obstacle value of 1 and 2 in the occupancy grid, the Manhattan grid is computed with the infinite for obstacle and filling up the empty space with number until the position of the vehicle is reached. We have obtained the path and path array matrix as shown below.

```
Occupancy Grid
[[ 0.  0.  0.  0.  0.  2.  2.  2.  2.  0.]
 [ 0.  0.  0.  0.  0.  2.  1.  1.  2.  0.]
 [ 0.  0.  0.  0.  0.  2.  1.  1.  2.  0.]
 [ 0.  0.  0.  0.  0.  2.  2.  2.  2.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  2.  2.  2.  2.  0.  0.  0.  0.  0.]
 [ 0.  2.  1.  1.  2.  0.  0.  0.  0.  0.]
 [ 0.  2.  1.  1.  2.  0.  0.  0.  0.  0.]
 [ 0.  2.  1.  1.  2.  0.  0.  0.  0.  3.]]

Manhatten Grid
[[ 18.  17.  16.  15.  14.  inf  inf  inf  inf   9.]
 [ 17.  16.  15.  14.  13.  inf  inf  inf  inf   8.]
 [ 16.  15.  14.  13.  12.  inf  inf  inf  inf   7.]
 [ 15.  14.  13.  12.  11.  inf  inf  inf  inf   6.]
 [ 14.  13.  12.  11.  10.   9.   8.   7.   6.   5.]
 [ 13.  12.  11.  10.   9.   8.   7.   6.   5.   4.]
 [ 14.  inf  inf  inf  inf   7.   6.   5.   4.   3.]
 [ 15.  inf  inf  inf  inf   6.   5.   4.   3.   2.]
 [ 16.  inf  inf  inf  inf   5.   4.   3.   2.   1.]
 [ 17.  inf  inf  inf  inf   4.   3.   2.   1.   0.]]
```

Figure 29: Occupancy Grid and Manhattan Grid Matrix

```
Path
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 1.  1.  1.  1.  1.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  1.  1.  1.  1.  1.]]
```

Figure 30: Path Generated

```
ath Array:
[9, 0], [8, 0], [7, 0], [6, 0], [5, 0], [5, 1], [5, 2], [5, 3], [5, 4], [5, 5], [6, 5], [7, 5], [8, 5], [9, 5], [9, 6], [9, 7], [9, 8], [9, 9]]
```

Figure 31: Path Array

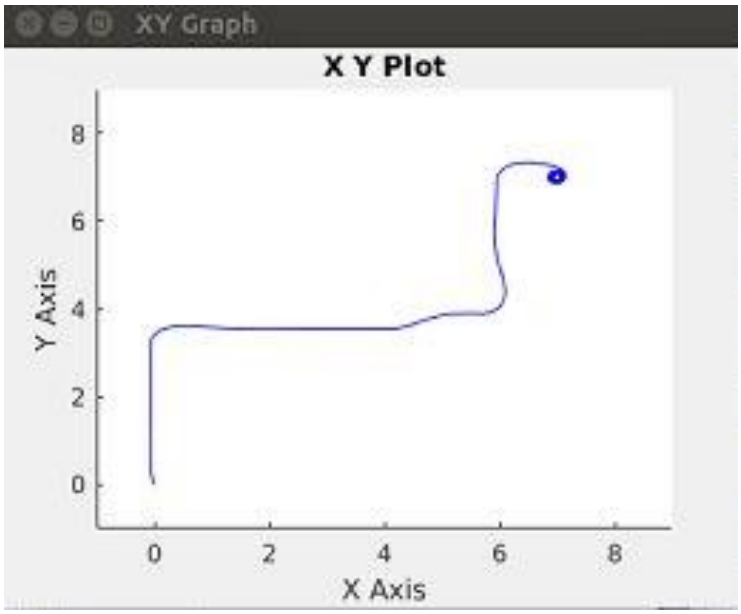

Figure 32: XY plot of Path

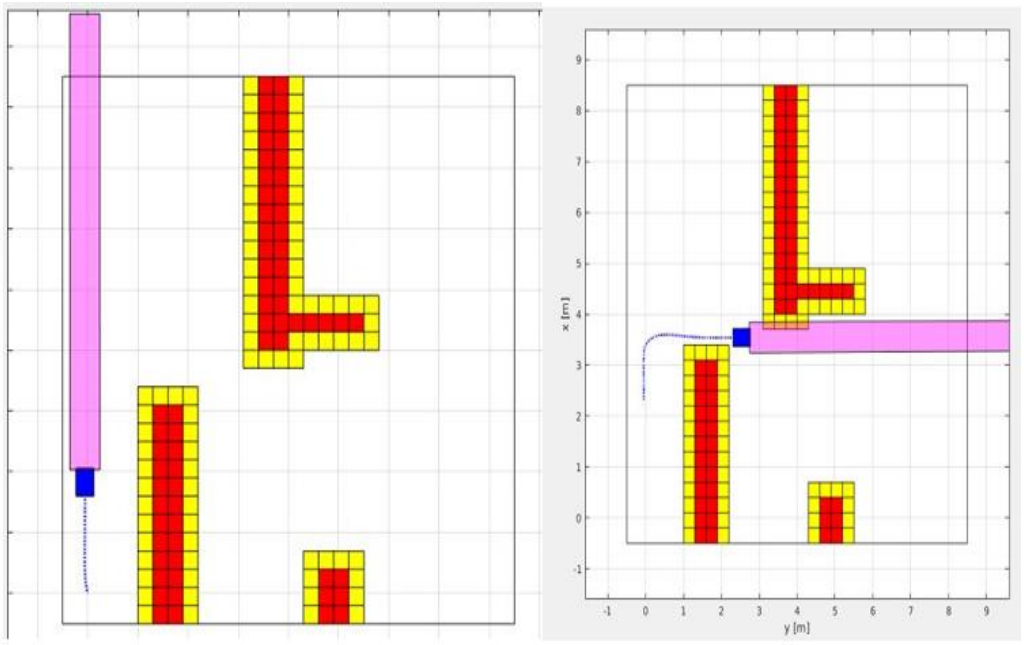

Figure 33: Path Planning Simulation

We were able to successfully integrate the Manhattan distance method and able to generate path for our vehicle in simulation, these method is implemented in our platform to move the vehicle into the path that is generated as shown in the simulation results.

15 EXPERIMENTS

We carried our first experiment trying to run the vehicle with goal at 7m with origin at (-0.5,-0.5), the vehicle was able to generate and reach approximately at 7m .We carried second experiment with goal at 7m at x-coordinate and 3m as y-coordinate run the vehicle the vehicle was able to generate path and waypoint to reach the goal, we continued with the experiment by placing the obstacle cell at 3m and 4m in a straight path. The platform behaved in a weird manner, the vehicle moved back for a while and start have a circular motion in a point and stop at that location without moving ahead. We figured it out that our experiment results are affected by the surface .We carried out some experiment inside the lab which has some rug and the vehicle was did not moved in a desired location because the friction between wheel and the surface. The dynamic test was carried out inside the atrium of Nedderman Hall with the path planner implemented and waypoint implemented by replacing the navigation block inside the sonar way point waypoint navigation Simulink. We first tried to experiment by avoiding the static avoidance and we were successful to perform the path planning. Occupancy grid was used to place the static obstacle and to give the goal location. We were able to perform the task very efficiently but we found that path executed with each number of experiment with same setting was not exact. Since vehicle wheel slips at different location and at different time which changes the speed it makes turn and move to reach the goal. Second experiment was carried out dynamic test, we carried out two approach for this test. We integrated vision model that was used previously for vision avoidance into the navigation block for changing the heading angle of the vehicle. The vehicle was able to generate

path just before the vison obstacle but once it saw the obstacle and tried to avoid it the vehicle lost it path and was not able to generate path to reach goal, . We found this reflexive method is not good to reach a goal with path that was generated because the map is required for the obstacle and must translate the obstacle in real frame into the occupancy grid. The second approach used was that is mapping the obstacle into occupancy grid by using the information from camera calibration, the vehicle was not able to reach the goal after it avoided the obstacles, the vehicle just made a path in circular motion, we could not find the solution for these and we were left with only static obstacle path planning for our path planning module.

16 Conclusion And Problem Encountered

16.1 Vison Avoidance and Waypoint Navigation

Our color detection and avoidance module was successful during our experiment but during our real demonstration our NUC shutdown for many reason and the connection with the platform was slow. We figured out the three was connection problem by the time we figured out and was ready for demonstration, we already lagged in time for demonstration on that day. We were just able to perform 2-3 test before the real demonstration and was not able to tune the vehicle properly due to time factor. We decided to go without tuning to perform the real demonstration and was able to demonstrate that it was able to avoid the blue color but was not successful with other waypoint navigation. Had we time for changing the parameter we would have successfully completed the demonstration.
The main difficulty that we encountered was the lag inherent in our control method: by the time we received an image to process, it was already out of date. For forward and backward movement, this did not present a problem since the image did not change much when moving forward or backward, but when turning, this presented a major problem. Reflections of the overhead fluorescent lights in particular caused the change in the pixel that would not happened if there was reflection so we decided to lower the pixel value in the programing for the avoidance. Due to lag by the time the algorithm detects the color our vehicle already run over the color; instead, we were forced to use the lower resolution and change in the program by changing the pixel value that was used for avoiding the color.

16.2 Path Planning Component

After having issue of PID tuning for the first demonstration of the vision part we had to tune the PID values before implementing the path planner. After tuning these parameter we carried out some of the experiment for path planning and the problem we encountered during our test is discussed briefly. Some of the issues we encountered during our testing is that the vehicle behaved differently every time we run the same experiment with the same configuration. We really could not address out these abnormal behavior of the result of the path executed by the vehicle. We also faced some challenges populating the grid. Some of the issues we encountered .When we update the obstacle into the map, the path planner thinks the vehicle thinks at (0, 0) location and it has already passed the obstacles for example the vehicle is at (5, 5) and sees an obstacle at (5, 7) then the obstacle is updated into the map at (0, 2).The second problem we encountered during our testing was when the vehicle sees the obstacle and update that into the world, the vehicle has already moved from the position it has mapped the world and the path plan was computed which would basically overshoot our vehicle faraway from goal location. Due to limitation of time to experiment we were not able to fully utilize the dynamic obstacle mapping for our demonstration. The vehicle lost it connection during our testing which hindered our

experiment to fully understand the problem that it to implement the mapping of dynamic obstacle and solution to it. So we were successful in avoiding the static obstacle and reaching the goal from the path that is calculated by our Manhattan distance path planner. We also figured it out that shortest path is not the fastest path. The path in which we can make less turn is the fastest path and we can increase the velocity of vehicle by moving vertical and horizontal path. We were able to attain good map of world and navigate them with some degree of precision with our Manhattan distance planning scheme. However several aspect of our project can me made better by improving the quality of map by using the other sensor to obtain the precise location and information of the object. We can use better algorithm for path planning .For example we can use A* method. Since camera is really slow it would have been better if we had used Matlab with open CV which can make process faster by processing less number of frame. Although this project seemed simple in theory, the actual implementation of it was more difficult than expected.

17 ACKNOWLEDGMENT

First of all, we are very thankful to Professor Dr. Wan for providing us lab and instrument during our project. Similarly, we are very thankful to TA Mr. Songwei for guiding and providing the missing components and new motors for making our project successful. We are very thankful to all the professor for giving us an opportunity to understand the unmanned vehicle system and challenge that need to be overcome for running the platform as desired.

18 REFERENCES

1. http://ardupilot.org/rover/docs/common-table-of-contents.html. accessed Feb 16 2017
2. https://www.pololu.com/category/124/roboclaw-motor-controllers. accessed Feb 22 2017
3. http://www.maxbotix.com/documents/XL-MaxSonar-EZ_Datasheet.pdf accessed Mar 18 2017