MULTI-SENSOR GUIDANCE OF AN AGRICULTURAL ROBOT

BY

HAO GAN

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Masters of Science in Agricultural & Biological Engineering
In the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Advisor:

Associate Professor Tony E. Grift

ABSTRACT

The increasing demand for high-density soil data, and the high labor cost associated with manual methods, have encouraged the development of autonomous alternatives. In this study, a mobile robot named 'AgTracker' was developed as a platform for an autonomous soil sampling machine. The robot, equipped with a low-accuracy GPS, a LIDAR scanner, an electronic compass visited human-defined locations through its auto-navigation system. This system also had a user-friendly interface, which enabled operators to set waypoints by clicking on Google Maps®. Locations could also be remotely monitored in real time through this interface. An Xbee wireless network was built to make the remote waypoints set up and monitoring possible.

The robot was tested on campus of University of Illinois at Urbana-Champaign. It could visit waypoints one by one successfully in most cases. The robot's localization errors, which were the distances between its true visited locations and set waypoints, were evaluated. An average error within 0.2 m was achieved.

To father, mother, and Hongping

ACKNOWLEDGEMENTS

First, I am very grateful to have Dr. Tony Grift as my advisor. He encouraged me to explore in the field of agricultural robotics and inspired me to try new ideas. With his expert guidance, I was able to learn and improve a lot on the road. This research would not have been possible without his support and guidance. I would also like to thank my committee members Dr. Sunil Mathanker and Dr. Krishnan, for their advice on improving my research and thesis. I wish I could have had more discussion with them so that I could accomplish more in the research. I would also like to thank Dr. Alan Hansen for his continuous guidance and support during my whole graduate study period. I was able to explore more aspects in the agricultural field with his help.

I am also very thankful to have many friends and colleagues in my department. I truly enjoyed working with Mr. Liang Tao, Dr. Haibo Huang, Mr. Robert Reis and Ms. Wei Zhao. Their assistance, cooperation, and friendship were important to my research life in the department.

Last but not least, I would like to express my thankfulness to many friends I met here in Champaign, Tyler Penn, Kellie Warren Penn, Gary Umphrey, Aaron Bird. Because of them, I could not only study with brilliant researchers, but also have a wonderful experience in learning the culture and enjoy my life in a different country.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS ..... **viii**

CHAPTER 1 INTRODUCTION **1**

1.1 RESEARCH MOTIVATION 1

1.2 OBJECTIVES 4

CHAPTER 2 LITERATURE REVIEW **5**

2.1 GPS LOCALIZATION FOR MOBILE ROBOTS 5

2.2 GPS-ALONE NAVIGATION 6

2.3 GPS-AUGMENTED NAVIGATION 6

2.3.1 Odometer Assisted GPS Navigation *7*

2.3.2 GPS Assisted Inertial Navigation System *7*

2.4 LIDAR BASED LOCALIZATION FOR MOBILE ROBOTS 8

2.4.1 Landmarks *9*

2.4.2 Scan matching *9*

2.4.3 Map building *10*

2.5 LITERATURE SUMMARY 10

CHAPTER 3 MATERIALS AND METHODS **12**

3.1 AGTRACKER PLATFORM 12

3.1.1 Drive train *13*

3.1.2 Electrical Motor Control Unit *13*

3.2 REMOTE CONTROL UNIT 14

3.2.1 Hardware *14*

3.2.2 Software *15*

3.3 AUTO-NAVIGATION SYSTEM 15

3.3.1 System Architecture *16*

3.3.2 GPS Control Unit *17*

3.3.3 XBee Wireless Communication *20*

3.3.4 LIDAR Control Unit *22*

3.3.5 Combination of GPS and LIDAR Control Unit *26*

3.3.6 Google Map Waypoint Control Unit *26*

3.3.7 Camera Triggering Unit *27*

CHAPTER 4 EXPERIMENTS AND RESULTS **28**

4.1 REMOTE CONTROL 28

4.2 GPS NAVIGATION 28

4.2.1 Data processing *28*

4.2.2 Experiment Design *30*

4.2.3 Results and Discussion *31*

4.3 LIDAR NAVIGATION 31

4.3.1 LIDAR Navigation Frame *32*

4.3.2 Data Processing *32*

4.3.3 Experiment Design *33*

4.3.4 Results and Discussion *33*

4.4 GPS AND LIDAR COMBINED NAVIGATION 36

4.4.1 Transformation between GPS navigation frame and LIDAR navigation frame *36*

4.4.2 Experiment Design *37*

4.4.3 Results and Discussion *38*

4.5 COMPARISON AMONG NAVIGATION SYSTEMS 39

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS **40**

5.1 SUMMARY 40

5.2 RECOMMENDATIONS FOR FUTURE WORK 41

5.2.1 Improving control algorithms *41*

5.2.2 Adding Inertial sensors *42*

5.2.3 Using vision-aided inertial navigation system *42*

5.2.4 Using wifi network *43*

5.2.5 Software implementation *43*

REFERENCES **44**

APPENDIX A: ARDUINO IMPLEMENTATION CODE **48**

APPENDIX B: MATLAB IMPLEMENTATION CODE **60**

LIDAR_GPS CONTROL.M 60

PLOT_GOOGLE MAP.M 65

GPS_INITILIZE 77

LIDAR_INITIALIZE 79

CONVERT_GPS_TO_LIDAR 82

GPS_TO_LIDAR 83

READ_GPSXBEE_NEW4 83

LIDAR_COMMUNICATION 85

LIDAR_READ 88

LIST OF ABBREVIATIONS

USDA	United States Department of Agriculture
NRCS	Natural Resources Conservation Service
GPS	Global Positioning System
GNSS	Global Navigation Satellite System
GLONASS	Global Navigation Satellite System
RTK	Real-Time Kinematic
INS	Inertial Navigation System
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
NMEA	National Marine Electronics Association
UIUC	University of Illinois at Urbana-Champaign
API	Application Programming Interface
URL	Uniform Resource Locator

CHAPTER 1 INTRODUCTION

The impetus to develop methods and techniques to collect agricultural data is increasing, because agricultural surveys have been used worldwide for estimating agricultural resources (Piersimoni & Bee, 2010). Among existing surveys in agriculture, soil sampling has been used frequently. As the basis of agricultural systems, soil provides critical resources such as nutrition and water for crops. The objective of soil sampling is to measure the average nutrition level of a field and the variability of nutrients of regions in a field (Richard & Gary, 1994). Based on field types and conditions, soil sampling methods can vary. Major sampling methods include judgmental sampling and random sampling. Both methods require expensive labor if the size of a field is large (Carter, 1993). Thus, automated sampling methods are necessary in order to make accurate and detailed soil maps. Autonomous robots which have been researched by many institutes are potentially ideal candidates for conducting auto-sampling tasks. An effective auto-sampling robot should consist of (1) an autonomous field navigation system (2) equipment for taking samples and (3) units for sample storage. Among those three requirements, developing an autonomous field navigation system is the most difficult because it requires the system to be able to deal with varying weather and field conditions. In this thesis, the development of an autonomous navigation robot which can be used in varying weather conditions and on flat fields is described.

1.1 Research Motivation

Soils are regarded as main contributors for food production, and the demand for real-time soil data is increasing dramatically. The resolutions of existing soil maps are very low, which is not helpful for most land management (Sanchez, et al., 2009, p. 325). Figure 1.1 and Figure 1.2 show the soil maps of Illinois, USA (USDA-NRCS, 2014). These maps provide users with general

information of local soil types and contents. However, to plan and manage individual farmlands, more precise and up-to-date information of each field is required.

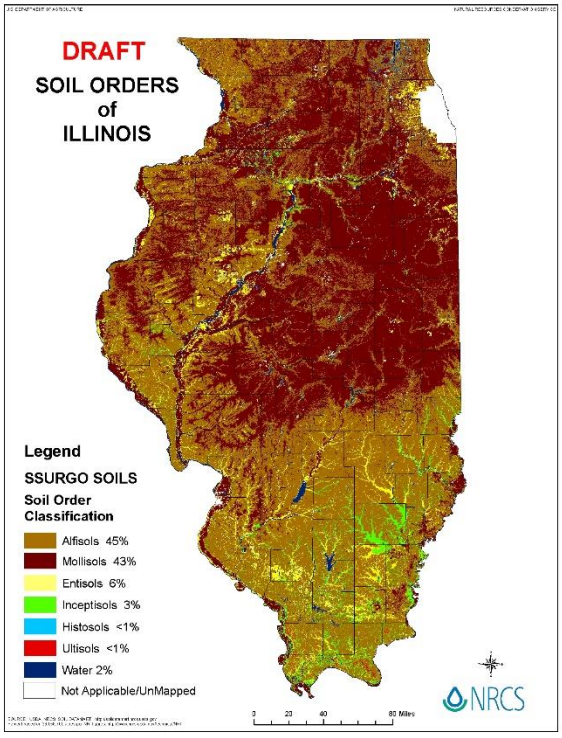

Figure 1.1 Brief Description of Classes

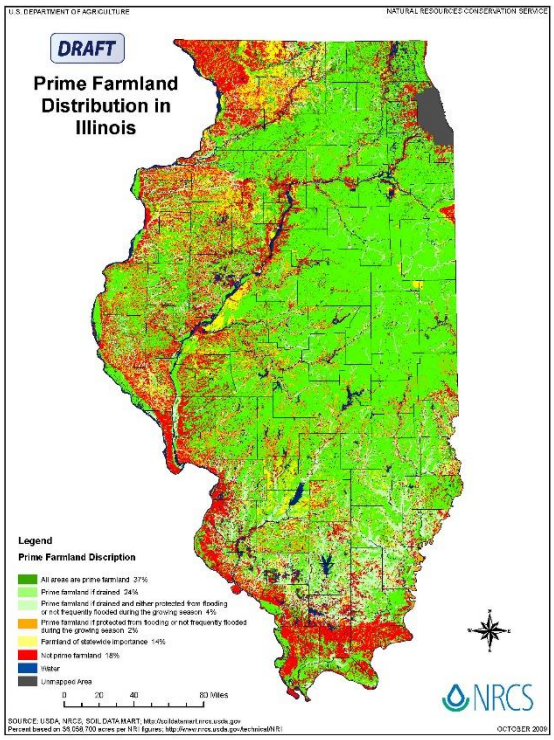

Figure 1.2 Prime Farmland

To obtain accurate soil data in each field, sampling soil is an essential step. Based on field conditions and usages, soil sampling can be divided into judgmental and random samplings. Judgmental sampling is a method in which sampling densities vary, based on observable conditions of different regions of farmlands. The accuracy of this method is hard to estimate because it depends on investigators' experience when planning sampling points. Random sampling, on the other hand, does not rely on investigators' judgments, but its accuracy is affected by sampling sizes. Intuitively, the larger the sampling size, the more accurate results can be expected. However, when the sampling size is large enough to a certain level, increasing sampling numbers in unit areas helps little in increasing accuracy. Based on Carter (1993), in most cases, 25 sampling points in 0.5 ha is the maximum, while 10 sampling points is the minimum. Figure 1.3

shows the comparison of soil maps with varying numbers of sampling sizes (Richard, 1994). In general, when the field size is large, the sampling size is always required to be large in order to build accurate soil maps.

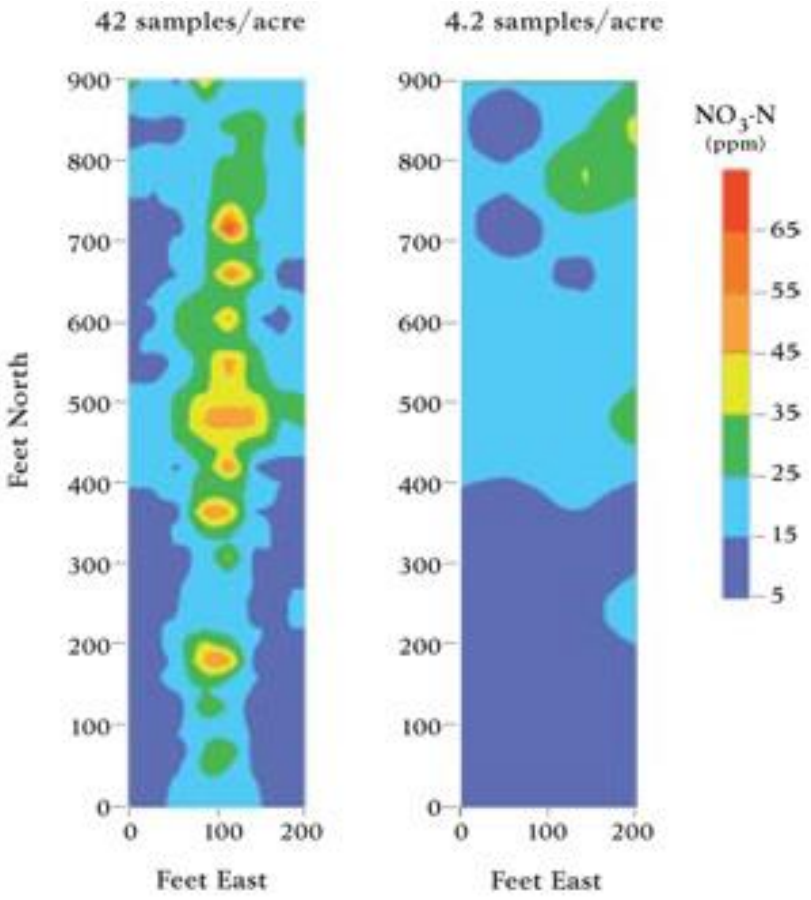

Figure 1.3 Comparison of built soil map with different sampling sizes

An autonomous sampling robot is desirable to relieve humans from expensive labor for collecting large numbers of soil samples. As mentioned previously, three stages are necessary for the robot to accomplish sampling tasks; The robot should firstly have an autonomous navigation system which can guide itself in fields to visit all sampling points. The navigation system has to be reliable under most weather and field conditions. Secondly, at each sampling point, the robot should be able to collect soil samples with proper tools. Finally, on the robot, there should be

enough space for separating and storing soil samples. Since there are already well designed soil sampling tools in markets, and designing of storage space is not difficult, developing a robot with a robust autonomous navigation system becomes the core for a sampling robot. Noticing that most farmlands in the mid-west of the United States are relatively flat, and to make the task easy in the first step, the developed robot was aimed at flat farmlands.

1.2 Objectives

This objective of this study is to develop a robot with the ability to navigate itself in flat fields. To be specific, the robot can be controlled remotely by an operator. The operator should be able to set locations of sampling points and monitor the positions of the robot. Besides, image data of soil at each sampling point are also desired in order to better estimate status of soils around sampling points. With the above requirements, the specific objectives of this study are listed as follows:

(1) Develop a robust positioning system with an accuracy better than 1 m;

(2) Build wireless communication network

(3) Add a Google map for setting up sampling points

(4) Create a real-time display of robot positions and trajectories

CHAPTER 2 LITERATURE REVIEW

This chapter illustrates research and techniques related to ground mobile robot navigation problems. Section 2.1 discusses various methods related to GPS on localization problems. Section 2.2 discusses the usage of LIDAR on navigating mobile robots.

2.1 GPS Localization for Mobile Robots

The Global Positioning System (GPS) is a widely used navigation system in North America. It is one of the three major Global Navigation Satellite Systems (GNSSs) in the world, the other two being GLONASS and Galileo, developed in Russia and Europe respectively (Farrell, 2008). The principle of GPS localization is to measure the distances between a GPS receiver and several satellites. This is achieved by multiplying the speed of signals which is the speed of light in principle, and measuring the signal's travel time from a satellite to the receiver. Errors occur in both the measurements of the speed of the signals and travelling time, which are caused by atmospheric interface, multipath interface and so on. Because of those errors, corrections methods have been developed depending types of received signals. There are two types of signal measurements used for GPS being pseudorange or code phase measurement and carrier phase measurement (Feuerbacher & Stoewer, 2006). The pseudorange measurement is the most common method which is for civil use while the carrier phase measurement is typically for military use because of its high price. Based on these two types of measurements, GPS navigation applications can be divided into GPS-alone navigation and GPS-Augmented navigation.

2.2 GPS-alone Navigation

Localization for mobile robots usually requires higher accuracy than automobile navigation in cities so that a high accuracy GPS is necessary if GPS is the only position localization tool. Among existing accurate GPSs, differential GPS (DGPS) and real-time kinematic GPS (RTK GPS) are mostly used for projects. Raible, Blaich & Bittel, (2010) developed a DGPS system equipped with two GPS receivers. The system worked with single frequency carrier phase observation and a Kalman Filter was used to estimate the system status. This system provided high accuracy in both static and dynamic states while keeping its price low. However, the limitation was that its accuracy was easily affected by weather. RTK GPS provides a higher accuracy than DGPS, it reaches centimeter level or sub-centimeter level. Sukkarieh, Nebot & Durrant-Whyte (1999) conducted a project on path tracking of vehicles for agricultural usage where vehicle sliding happened frequently. An RTK GPS was used to provide accurate positioning while a model was built to estimate the vehicle's sliding magnitude. The major errors in the path tracking task were due to the sliding estimation model while the RTK GPS had a satisfactory performance, but at a relatively high price.

There has been ample research on improving GPSs' accuracy. However, GPS-alone navigation for mobile robots was not very popular because neither DGPS nor RTK GPS could meet the basic requirements for most robots, being robust and low-cost. Thus, GPS for robot guidance is typically augmented with additional sensors in a sensor fusion arrangement.

2.3 GPS-Augmented Navigation

GPS-Augmented navigation is a method in which multiple sensors are added to provide guidance information together with GPS. The most common GPS-Augmented navigation is dead reckoning which uses an odometer or inertial sensors.

2.3.1 Odometer Assisted GPS Navigation

A GPS unit has to receive signals from at least four satellites which have to be in the line of sight of the receiver. A GPS may have poor performance or lose its position when the sky is blocked or partially blocked by tall objects. That problem could happen when a robot is running in places such as city's urban canyons and streets with high buildings surrounding it (Geier, 1996). An odometer can assist the GPS and provide a mobile robot with location information for a short time when the GPS loses full connection with satellites. Ohno, Tsubouchi, Shigematsu, & Yuta (2004) combined a DGPS with an odometer to test the accuracy of robot navigation when GPS signal was poor. This method not only helped with robot localization when satellites could not be viewed directly from GPS, but also to deal with GPS multi-path errors, where the signal from a satellite does not follow a straight path to the receivers, but reflects off objects such as tall buildings. However, the method suffered from incremental errors caused by wheel slip, a common limitation in most odometer assisted GPS navigation. Thus, to obtain more reliable and accurate navigation information, more advanced methods are necessary.

2.3.2 GPS Assisted Inertial Navigation System

Inertial Navigation System (INS) which depend on mechanics laws usually contain three accelerometers and three gyroscopes. By combining these two sets of measurements, theoretically, the system is able to calculate its position (Titterton & Weston, 2004). However, due to the bias of INS alignment and errors in accelerometers and gyroscopes, additional information of position or attitudes of the system are essential, which can be provided by a low-cost low accuracy GPS unit.

Most GPS assisted INS applications contain an IMU and a low cost GPS unit. The major difference among existing GPS/INS applications lies in their method of data fusion. Particularly,

various types of Kalman Filter were applied to fuse IMU and GPS data. Sukkarieh, et al. (1999) developed a high integrity navigation system in which a strap-down IMU was aided by either a standard or carrier phase GPS. A standard Kalman Filter was used to detect errors in both GPS and IMU, whereas status such as positions, velocities, and attitude were estimated in real-time. The accuracy of the system depended upon the accuracy of GPS and the alignment process that estimates the vehicle's initial status. Because a standard Kalman Filter was not capable of estimating the status with high accuracy, more research was done with advanced Kalman Filter techniques. Qi & Moore (2002) created a direct Kalman Filter approach to fuse GPS and INS where nonlinearities from both sensors were processed prior to use by the Kalman Filter. Sasiadek & Wang (1999) used a fuzzy Kalman Filter to fuse simulated IMU and GPS signals. Based on simulated data, the positioning accuracy was improved, though no field experiment was done. Besides using an advanced Kalman Filter, some researchers added more sensors to help with status estimation. For instance, Zhang, Gu, Milios, & Huynh (2005) added a digital compass to the traditional IMU/GPS system to provide it with high frequency heading measurements. Those measurements together with GPS measurements were used for status corrections. The correction process was implemented by an extended Kalman Filter in which vehicle's status could be accurately estimated. There were many other applications of IMU/GPS systems that used Kalman Filters, they mainly varied in the type of Kalman Filter used.

2.4 LIDAR Based Localization for Mobile Robots

Light Detection and Ranging (LIDAR) technology has been used widely in navigating mobile robots. Existing LIDAR related navigation methods for robots can be divided into three major categories, although these robots could have very different applications. Those three categories are methods related to 1) landmarks, 2) scan matching, and 3) map building.

2.4.1 Landmarks

Landmarks, either natural or artificial, are observable features that a robot can recognize from a wide range of locations using its sensors (Borenstein, Everett, Feng, & Wehe, 1997). The idea of navigating using landmarks is simple: In a known environment, a robot measures its relative position to a landmark and updates its position information based on a known map (Núñez, Vázquez-Martín, Del Toro, Bandera, & Sandoval, 2008). Many researchers have used this idea to guide robots: Roumeliotis & Bekey, (2000) used features such as walls and corners as landmarks to navigate a robot in an indoor environment. Trees have also been used as landmarks for outdoor navigation (Zhang, Xie, & Adams). There were also some novel ideas on using landmarks. Kurazume & Hirose (1998) developed a cooperative method, in which several robots moved based on each other's positions. In that process, one robot stopped as a landmark and other robots moved based on that stationary robot's position. Next, a second robot stopped as a new landmark and all other robots moved based on the position of that second one. Robots continued this process until they all reached their desired positions.

Using landmarks is simple method of navigation, because it does not need many calculations and post processing. However, to use this method, a map needs to be known in advance. Because of this limitation, the landmark method is not suitable in many situations.

2.4.2 Scan matching

Scan matching is a process where two range scans are compared in order to find the translation and rotation of the second scan relative to the first one. Gutmann, Weigel, & Nebel (2001) developed a method for self-location of soccer robots using scan matching. They used a line matching algorithm to extract line segments from a scan and matched them with an *a priori* known map. This method can be extended into any polygon shaped field, but only in a small scale

environment. Diosi & Kleeman (2004) fused sonar and laser range finder data to generate a method for simultaneous localization and mapping. Sonar and laser readings were combined in this method. Point features from sonar reading were used together with line segments from a laser range finder. In this way, errors due to measurements from a single sensor can be removed effectively. Scan matching is fast and does not require expensive post-processing calculations. However, to use this method the surrounding environment must contain detectable features.

2.4.3 Map building

Rencken (1993) defined the map building problem as "Given the robot's position and a set of measurements, what are the sensors seeing?" As the "sensor", a LIDAR was usually mounted on a robot to explore and scan in an unknown environment. A computer built a map of the environment based on the scans. Fu, Liu, Gao, & Gai (2007) used a laser range finder and a camera to build a 2D map and vertical edges of an indoor environment. Surmann, Lingemann, Nüchter, & Hertzberg (2001) constructed a 3D range finder based on a 2D range finder and obtained a 3D map of an indoor environment. Cole & Newman (2006) used a similar method to build a 3D map of an outdoor environment. The method of map building gives more information of the environment and offers more options for the robot to perform different tasks based on built 2D or 3D maps. However, this method usually requires more sensors and more expensive post processing.

2.5 Literature summary

Typical navigation methods associated with GPS and LIDAR in literature were presented. Low-cost GPS, although widely used as a navigation tool, is limited by weather and environmental conditions. Therefore, to provide accurate positioning, it needs to be aided by reliable high rate sensors such as accelerometers and gyroscopes. LIDAR can measure distances of objects relative to itself with a high accuracy in a relatively short range, while the accuracy is inversely

proportional to the range of an object. In addition, objects must be distinct in shape from their surroundings. Thus, its usage was limited to short range detections, scan matching, and map building. By studying features of both GPS and LIDAR, and considering the conditions of fields where a robot would be traversing, a combination of GPS and LIDAR was tested in this project.

CHAPTER 3 MATERIALS AND METHODS

This chapter has three sections, which will cover most materials and methods used in the project. In section 3.1, the AgTracker platform, which was the base of the robot, will be briefly introduced. Section 3.2 will be focused on a remote control unit that required real time human operation. The auto-control unit was the most important part of this robot and will be described in section 3.3. Equipment and methods involved in more than one sections or units will be discussed only once when they are firstly mentioned.

In this project, the aim was to navigate a robot in a farm field which has following features: long range – about 50 by 90 square meters; flat – no obstacles in this field; uniform environment – absence of landmarks – no distinct object can be seen in the whole field. By analyzing all the features above and all existing methods, the decision was made to choose a method similar to using a landmark to approach the task. However, it has significant differences because the landmark in our method is moving and the laser range finder is stationary. In this research, an agricultural robot named "AgTracker" was used as the basis for navigation experiments.

3.1 AgTracker platform

The AgTracker platform is a simple, stable robot (Grift & Kasten, 2005; Grift, Kondo, & Ting, 2008; Xue, Zhang, & Grift, 2012; Bac, Grift, & Menezes, 2011; Xue & Grift, 2011). It features skid steering which allows the robot to move forward/backward, turn and spin easily. It mainly consisted of a drive train, batteries, and two DC motor controller boards. The original control hardware consisted of an obsolete BasicAtom microcontroller, which was replaced with a modern Arduino controller.

3.1.1 Drive train

The drive train of AgTracker features two motors (Astroflight 940P Geared Motor), from which power was geared down to four wheels as shown in figure 3.1. The transmission ratio between the motors and the wheels was 18:1. One advantage of this system is that it features two drive motors, which control the left and right wheels independently. This enabled AgTracker to spin turn even on a soil surface.

Figure 3.1 AgTracker drive train

3.1.2 Electrical Motor Control Unit

The electrical motor control unit comprised a custom made controller board and two motor controller boards (see figures 3.2 and 3.3). The custom made controller board functioned as an interface between a BasicAtom microcontroller and two motor controller boards. Its main usage for this project was to generate PWM signals from microcontroller to motor controller boards, which then controlled the speed and direction of the drive motors.

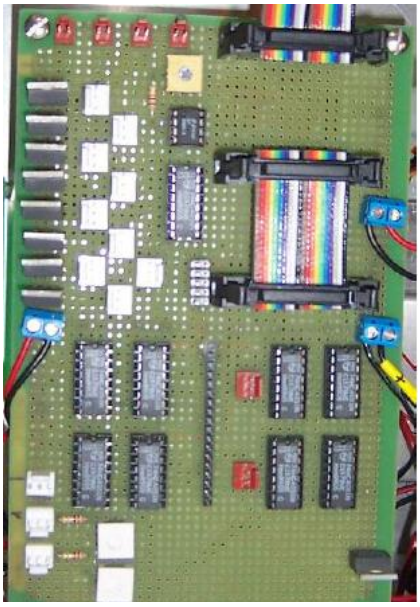

Figure 3.2 custom made controller board

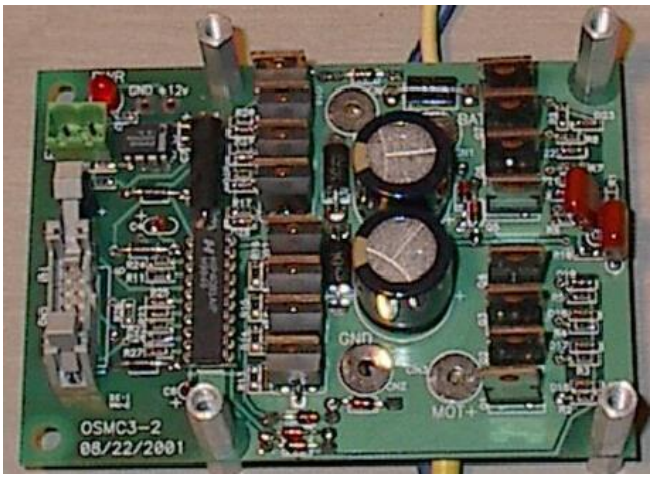

Figure 3.3 motor controller board

3.2 Remote Control Unit

Because the remote control unit was stable, it was built to control the robot manually in cases where auto-control was not necessary or did not work. This unit was insensitive to surrounding environment and would work well as long as its two components – transmitter and receiver, were in range.

3.2.1 Hardware

The robot was remote controlled using a 4 channel 72 MHz unit (Tower Hobbies 3000), with a range of approximately 500 meter. Since the robot was steered using skid steering, it only required two channels. Figure 3.4 shows the transmitter for this unit.

Figure 3.4 transmitter of tower hobbies system 3000

3.2.2 Software

The code that converts the analog signals from the remote control receiver into digital signals, was written in C/C++ within a software development kit (Arduino 1.0.5 R2 for Windows, Arduino SA). The digital signals were sent to the motor controller board to control wheels' speeds. Implementation codes can be found in Appendix A.

3.3 Auto-Navigation System

The Auto-Navigation System comprised a feedback network, containing a GPS unit, LIDAR, wireless communication network, Google map waypoint control unit, real-time monitoring, and camera control unit. Figure 3.5 shows how these units worked as a system.

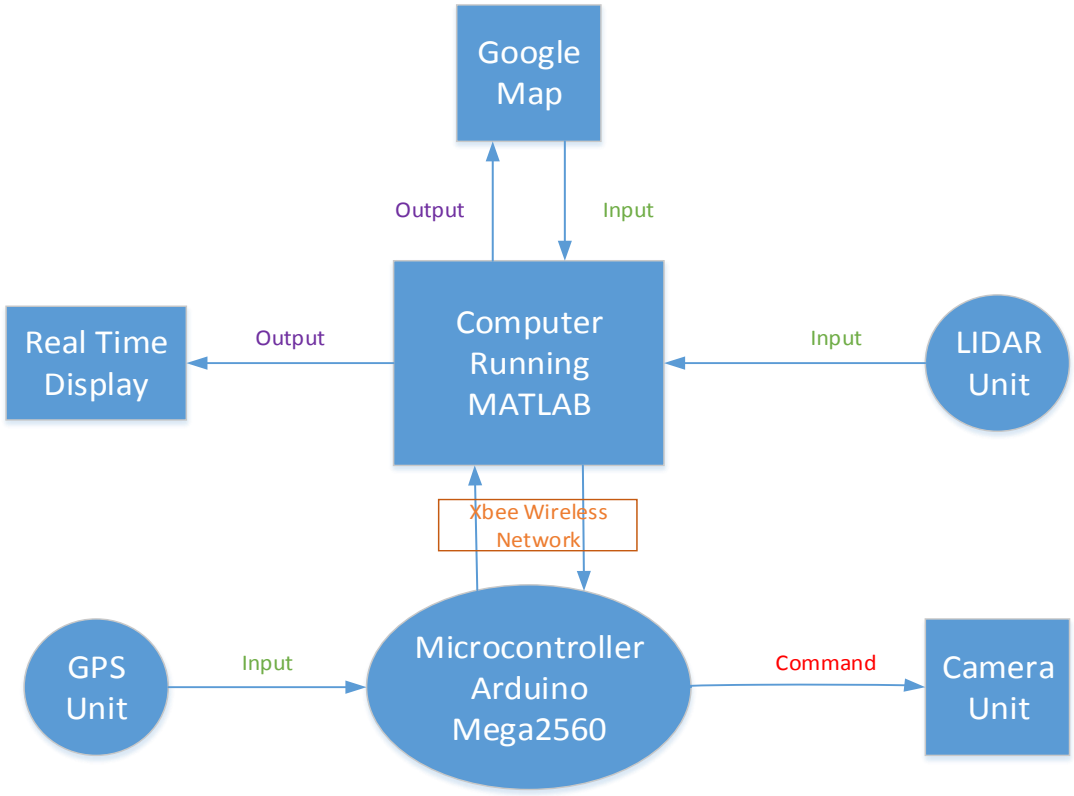

Figure 3.5 Auto-navigation system

To navigate the robot to a desired waypoint, first, a map of a user selected area was plotted, using Google Maps ® functionality (see Section 3.3.6). Waypoints can be selected by either clicking on the map or using pre-defined locations. Secondly, based on present location of the robot and waypoint locations, the system would drive the robot to visit waypoints one by one based on the input order, using GPS signals and LIDAR signals. At last, at each waypoint, the camera control unit would trigger a facedown camera in the front of the robot to take pictures. This process would not stop until the robot had visited all waypoints.

3.3.1 System Architecture

The auto-navigation system was similar to a human system which is made of brain, sensory organs, and the nervous system. The computer had all the intelligence, and it is given a task which

was to visit waypoints in this project. Next it analyzes signals from all sensors including GPS, LIDAR, compass and web-based google map and, finally, send commands back to sensors. All the sensors would receive signals in real time and transmit them to the computer. The bridge between the computer and sensors were the internet and an XBee wireless network (See Section 3.3.3).

3.3.2 GPS Control Unit

The GPS unit was one of the two independent navigation systems of the robot while the other one was LIDAR control unit. It had the ability to guide the robot from one point to another by itself when GPS signals were unavailable.

3.3.2.1 Hardware

The GPS Control Unit's guidance relied on a GPS receiver, an electronic compass, two controllers, a microcontroller and a laptop computer. A GARMIN GPS V (Figure 3.6) was used to receive satellite signals. A two dimensional position (latitude and longitude) was calculated when at least three satellites were locked onto and a three dimensional position (latitude, longitude and altitude) was calculated if at least four satellites signals were received. The position accuracy of the GPS was about nineteen feet on clear days when WAAS, a system of satellites and ground reference stations that improves position accuracy, was enabled. A standard NMEA sentence would then be sent every two seconds from the receiver to a computer.

Figure 3.6 Garmin GPS V

A three-axis electronic compass was used to determine the robot's heading (PARALLAX HMC5883L). It measured the Earth's magnetic field and output three values representing magnetic strength of three orientations X, Y, Z, as shown in Figure 3.7. One of the product's three standard mounting orientations was chosen, where the X-axis was the forward reference direction, Y-axis was the other horizontal reference direction and Z-axis was the vertical reference direction. The true heading of the compass can be calculated using only values of X-axis and Y-axis with the knowledge of local declination angle which can be checked online. The calculation process is shown below. The maximum output rate is 160 Hz with an accuracy of 1 to 2 degrees.

Direction (y>0) = 90 - [arcTAN(x/y)]*180/π

Direction (y<0) = 270 - [arcTAN(x/y)]*180/π

Direction (y=0, x<0) = 180.0

Direction (y=0, x>0) = 0.0

Heading = Direction ± Declination angle

(Honeywell AN-203)

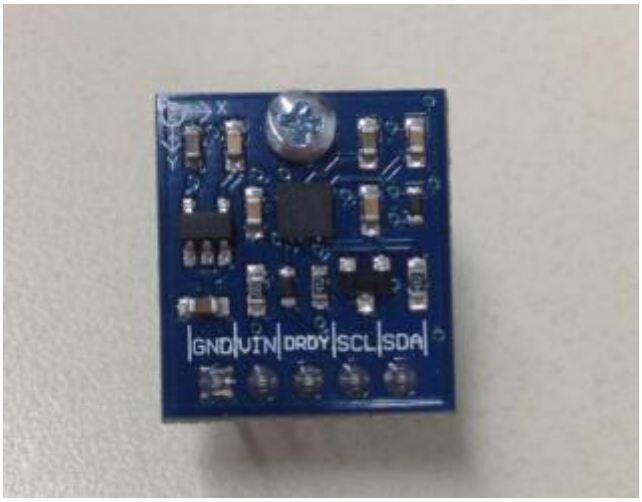

Figure 3.7 PARALLAX three axis HMC5883L electronic compass

The microcontroller used in this research was the Arduino Mega 2560 (Figure 3.8), which features 16 analog input pins, 54 digital input/output pins, a pair of I^2C pins and 4 pairs of hardware serial pins. Two I^2C pins (20 and 21) were used to receive compass values. A pair of serial pins (pin 3 and pin 4) were used to receive GPS sentences. Another pair of serial pins (pin 5 and pin 6) was used for XBee wireless communication.

Figure 3.8 Arduino Mega 2560 controller

A DELL VOSTRO laptop was selected to execute auto-control operations using MATLAB R2013b. It was equipped with an Intel(R) Core(TM) i5 CPU and 3.0 GB RAM. It had a 64-bit

operating system running under Windows 7 Enterprise. It also had three USB ports, where two of them were used to receive sensors' data.

3.3.2.2 Software

Arduino 1.0.5 R2 and MATLAB R2013b were the two major software packages used for this unit. Note that code was written in Arduino 1.0.5 R2, downloaded into Arduino mega 2560 microcontroller and executed continuously when the microcontroller was powered up. Detailed implementation code is in Appendix i.

3.3.2.3 Control Loop

Location and heading signals from the GPS control unit and compass were firstly received by the Arduino microcontroller. Next, the Arduino controller translates the signals, organizes them into one sentence, and transmits this sentence to the computer through the XBee wireless module. Simultaneously, the MATLAB program compares the true heading of the robot with the desired heading calculated by connecting the robot's current location and its target location. From this comparison, MATLAB's program calculates the required speeds of the left and right wheels. The speed values were sent back to the microcontroller at the end for controlling wheels' speeds. This was a continuous process before the robot arrived at its destination.

3.3.3 XBee Wireless Communication

A pair of Digi XBee Pro RF modules and Arduino compatible XBee shields were used to implement wireless communication in this project. The communication was built by pairing two XBee modules manually through software (X-CTU). To pair the modules, they must be in the same network and channel, which were set by ID and CH in modem configuration. The setting of destination addresses (DH and DL parameter in modem configuration) determined whether this two modules could communicate. Their settings are described as follows:

If DH of XBee1 is set as A and DL of XBee1 is set as B, then for XBee2, its DH should be B and DL should be A. A and B represent numbers between 0 and 0×FFFF. Figure 3.9 shows the main modifications to the XBees' modem configuration. The Arduino Xbee shield allowed data to be sent and received wirelessly between the serial ports on the Arduino Mega2560 and a computer. As Figure 3.10 shows, XBee1 was connected to the Mega2560 through the XBee shield and XBee2 was connected to computer through a USB port. The communication was fast and stable and its range could be as far as 1 mile outdoors and 300 feet indoors.

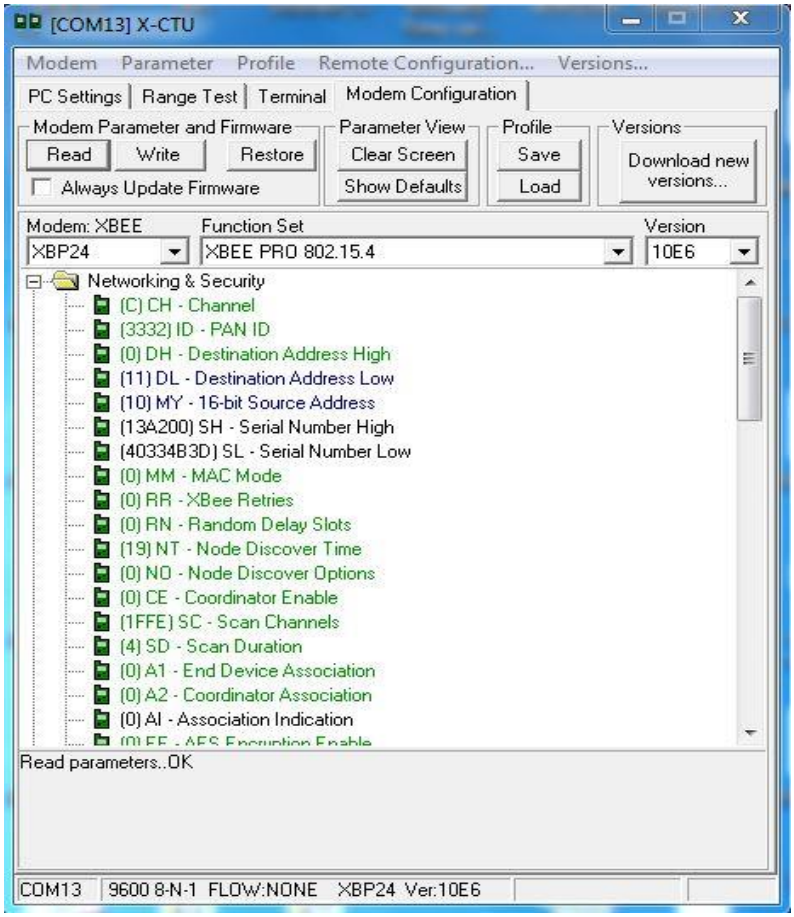

Figure 3.9 Major modifications in Xbee's modem configuration

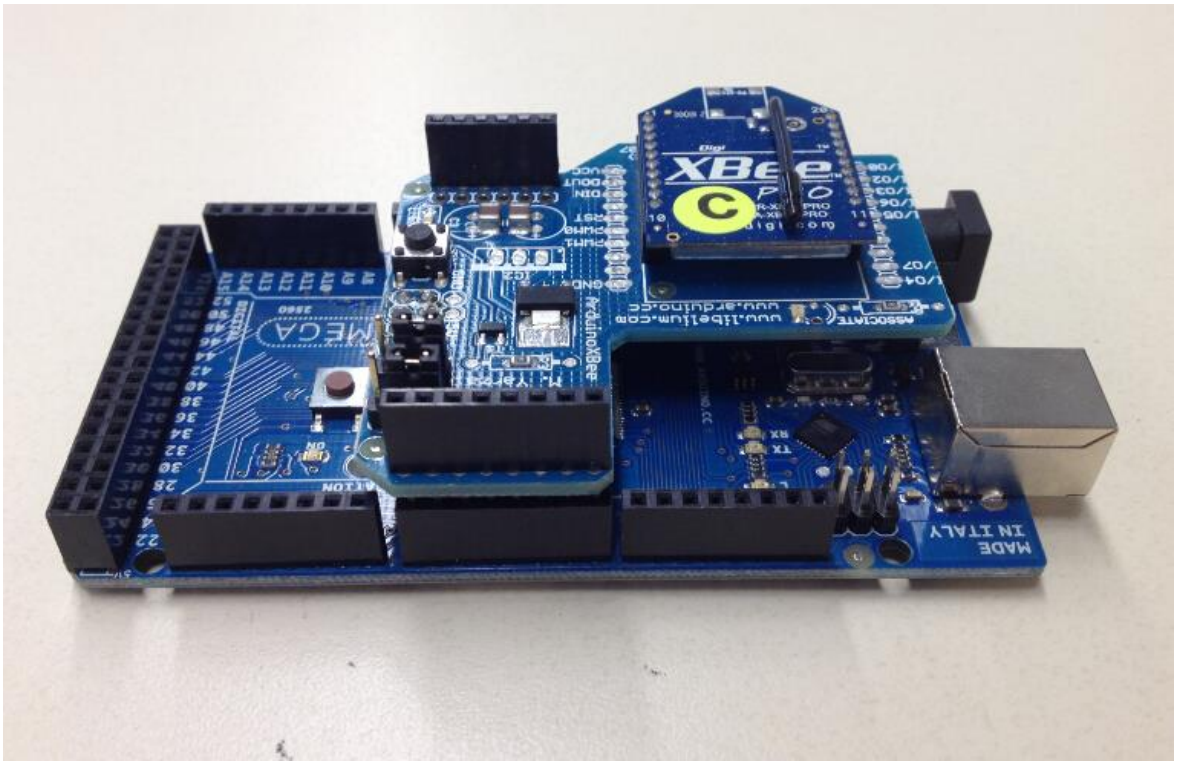

Figure 3.10 Xbee connected to Arduino Mega 2560 through an Xbee shield

3.3.4 LIDAR Control Unit

The LIDAR Control unit as mentioned before, is capable of guiding the robot to destinations by itself. It was more accurate and stable than the GPS control unit, but it has major limitations such as having a short range and requiring flat fields.

3.3.4.1 Hardware

The system consisted of a computer, a SICK LIDAR LMS 291, a PARALLAX three axis HMC5883L electronic compass and two 12 volt batteries. There was no specific requirement for the computer and batteries as long as the computer could run MATLAB R2013b and the batteries could provide 24 volt. In this unit, the same DELL VOSTRO laptop computer was used to execute tasks. The compass was the same as introduced in the GPS control unit. The LIDAR LMS 291 was the key part of this unit. It measured the travel time of a laser pulse between the LIDAR and any object around it. There are five range and accuracy modes as listed below (© SICK AG · Auto

Ident · Germany). The selection of ranges and modes was based on the needs of our study and will be discussed in section 4.3.3. The location of the LMS needed to be adjustable. A custom frame allowed for mounting the LIDAR unit on any other structures (Figure 3.11). Since the location of the LIDAR was fixed, communication between it and the computer was implemented using a 5 meter long cable with an RS232 connector. A RS232 – USB converter was used to interface the serial output from the LIDAR with the laptop computer.

Table 3.1 LIDAR mode and angular range options

Angular Range	Angular Resolution	Number of Data Values
0˚ - 100˚	1˚	101
0˚ - 100˚	0.5˚	201
0˚ - 100˚	0.25˚	401
0˚ - 180˚	1˚	181
0˚ - 180˚	0.5˚	361

Mode	Measurement/Detection Range
mm Mode	0 - 8191mm = 8.191m
cm Mode	0 - 8191cm = 81.91m

Figure 3.11 Modifications to the LIDAR unit

3.3.4.2 Software

The control software of the LIDAR unit was written in MATLAB R2013b (MathWorks, Natick, Massachusetts). The procedure of communication setup and control commands were based on Quick Manual for LMS Communication Setup version 1.1. All implementation code is listed in Appendix i. To test the status of the LIDAR unit, an existing program was used, which provided real time display of objects' locations (Figure 3.12).

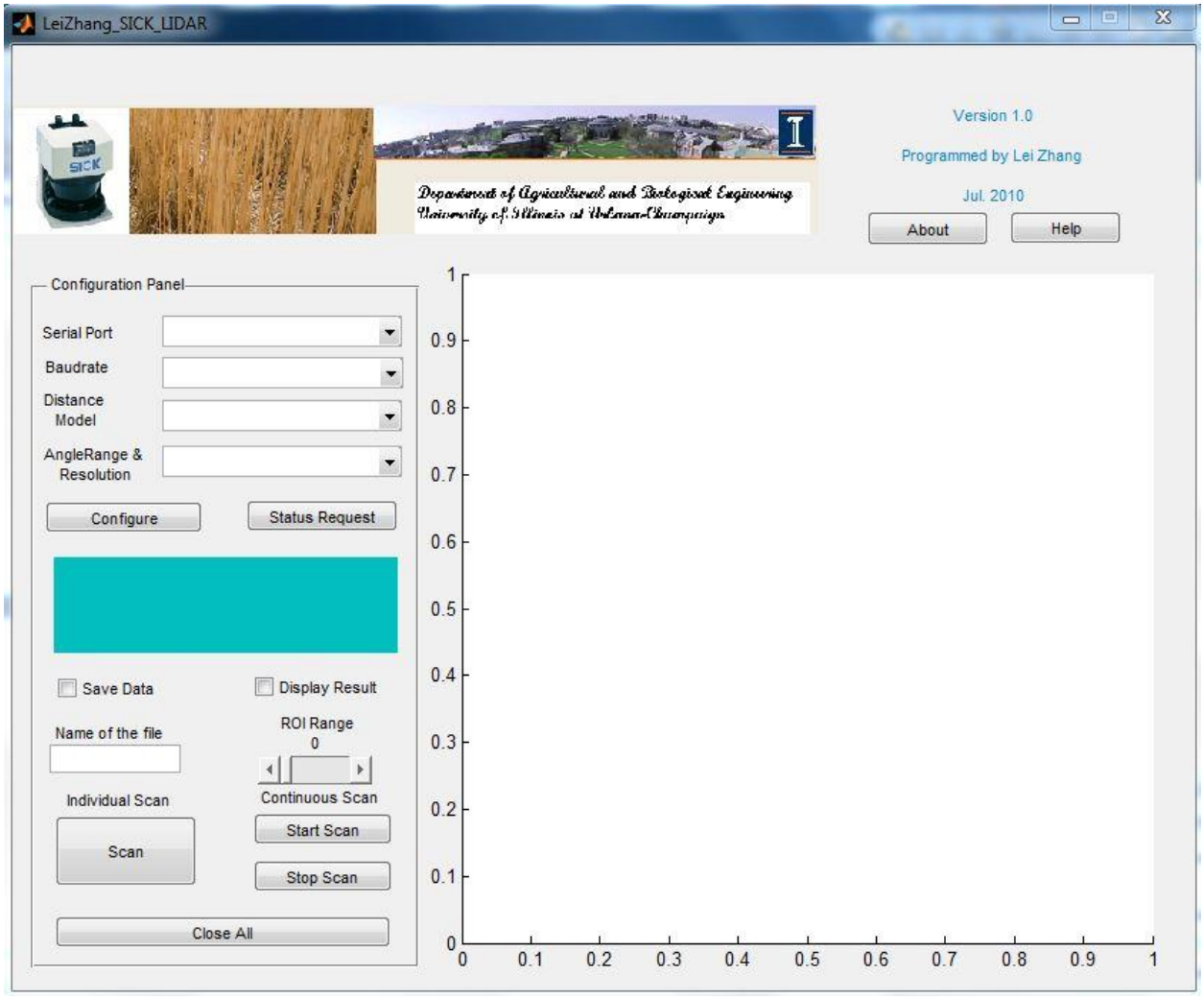

Figure 3.12 MATLAB GUI interface for LIDAR data acquisition (courtesy of Dr. Lei Zhang, UIUC)

3.3.4.3 Control Loop

The structure of the LIDAR control unit is similar to that of the GPS control unit. The LIDAR would detect the relative position of the robot with respect to itself and compare it with the destination. Note that because the LIDAR would scan any objects in its range, the program needed to filter all stationary objects by collecting many sets of data and remove those positions that remained constant, leaving the only moving object as the robot. Next, a targeted moving direction would be determined by calculation. That direction would be compared again with the robot's true heading provided by the electronic compass through the XBee wireless module. Based

on the above information, the program would calculate the required left and right wheel speeds and finally send them back to Arduino microcontroller. This control loop continued until the robot reached all of its target waypoints.

3.3.5 Combination of GPS and LIDAR Control Unit

While both the GPS and LIDAR control units were able to guide the robot individually, integration of these units offered a faster, more accurate and reliable method. The integration comprised only software adaptations, which allows for reverting back to single sensor if the other would fail, or considered redundant. After integration, the program executed the following steps.

Firstly, an initial robot position based on GPS was acquired by averaging 10 GPS data points. The initial position was recorded as a [Lat0, Lon0] pair. Secondly, an initial position was acquired from the LIDAR. The same initial position recorded by LIDAR was marked as a [X0, Y0] pair. Both initial positions were obtained while keeping the robot stationary. Thirdly, an algorithm was generated to calculate a one on one match between [Lat0, Lon0] and [X0, Y0]. This algorithm would be used throughout the process to match the GPS position with the LIDAR position received at the same time. Note that since LIDAR was more accurate than GPS, if the GPS and LIDAR positions did not match, the program would rely on the LIDAR generated positions. Finally, by analyzing the robot's current location and its destination, the program would generate a solution to guiding its next-step movement. If the location information from only one unit, GPS or LIDAR, was received, the program would use it to guide the robot.

3.3.6 Google Map Waypoint Control Unit

The Google map waypoint control unit was a user interface created using the Google map image API, a web service developed by Google. It enables users to create map images on a web page by sending URL parameters through a standard HTTP request (Static Maps API V2

Developer Guide, 2014). To display map images in MATLAB rather than on a web page, a MATLAB function created by Zohar Bar (2013) was applied. This function would take geographic locations, latitude, and longitude as input variables and display a google map image covering those locations. The waypoint control interface then enabled users to set waypoints by clicking on the displayed map or by manually entering precise latitude and longitude pairs.

3.3.7 Camera Triggering Unit

To obtain images of the area in front of the robot, an iPhone 4 with a camera trigger mechanism was used. The mounting height of the iPhone 4 was 45 cm above ground in which case its camera would take a picture of a 60 cm by 40 cm rectangular area when triggered. The iPhone 4 was inserted into a 3D printed case, mounted on a frame protruding from the front of the robot. The triggering of the iPhone was achieved by pushing one of its buttons with a solenoid. Since for proper action, the solenoid requires a rather large current which cannot be supplied by the Arduino Mega 2560's digital pins, a solid state relay (SSR) was used that only required an input in the range from 3 to 32V. A second advantage of the SSR was that it provides galvanic isolation through opto-coupling. When the robot arrived at each waypoint, the MATLAB program sent a signal to set the digital pin to high and the iPhone would take pictures.

CHAPTER 4 EXPERIMENTS AND RESULTS

In this chapter, experiment design and evaluation of AgTracker's performance are presented. Performance of remote control unit is shown in section 4.1. Section 4.2 and section 4.3 discuss GPS and LIDAR guidance respectively. In section 4.4, the combination of GPS and LIDAR guidance and its evaluation are presented. Section 4.5 elaborates on the Google Map API interface.

4.1 Remote Control

As mentioned in chapter 3, the remote control unit allowed for manual robot manipulation in cases where the auto-control system was not in use, for instance, to move the robot from and indoor location to an outdoor test field. Because the task was simple, the requirements of the remote control unit's range and sensitivity was not high. However, it had to be robust to ensure safety. The unit's robustness was tested before the auto-control system was completed and during eleven months of testing, it never failed, indicating that the system was robust and reliable.

4.2 GPS Navigation

GPS navigation was tested and the results were evaluated for comparison with LIDAR and GPS/LIDAR navigation later in section 4.5.

4.2.1 Data processing

To guide the robot, the GPS navigation unit received information from the Garmin GPS V and the PARALLAX electronic compass. The GPS provided location data in latitude/longitude pairs and the compass transmitted heading data in degrees. The data transmitting frequency of the GPS and compass were 0.5 Hz and 20 Hz respectively, whereas the maximum transmitting

frequency of the compass was 160 Hz. However, due to the low speed of the robot, 20 Hz was sufficient which saved memory space and improved processing speed. Since the update frequency of headings was much higher than that of locations, the system would regard the location of the robot to be constant before the next GPS data was received while controlling the speed of its wheels solely based on the updated heading information. This was a reasonable assumption, because the calculated direction of the line which connected the target location and the robot's location would not change much in a short time, especially when the robot is far from its target waypoint. When the robot moved closer to its target, the system slowed down the travel speed of the robot which alleviated the problem of low frequency data updates from the GPS. The change in target direction when the robot is far from its destination is shown in Figure 4.1, and when the robot is near its destination in Figure 4.2.

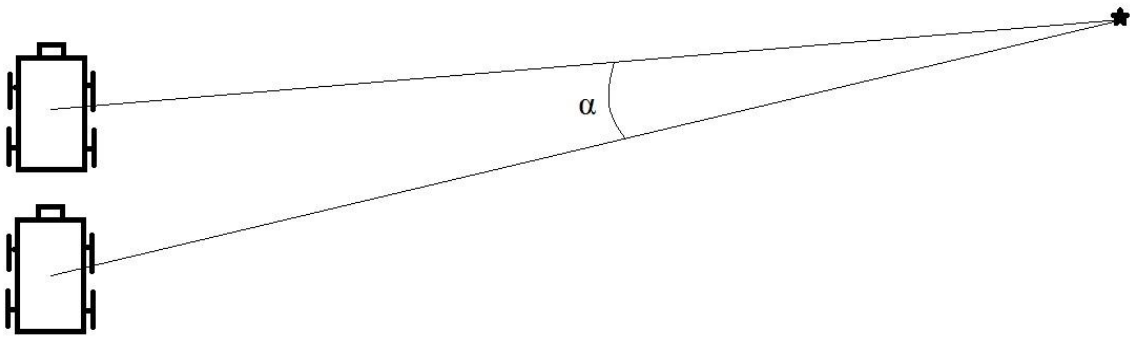

Figure 4.1 target direction change of α in a short time period when the robot is far from its destination

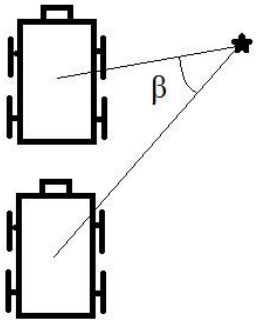

Figure 4.2 target direction change of β in a short time period when the robot is near its destination

4.2.2 Experiment Design

The experiment design for GPS navigation involved two factors, 1) selection of the experiment location and 2) selection of waypoints. The tests took place at the south quad of University of Illinois at Urbana-Champaign (Figure 4.3). The south quad is a flat lawn with a dimension of about 75 m × 120 m. Trees were present, but they were all located close to the edge of the field, and did not obstruct the experiments. The robot was driven to an arbitrary location on the quad under remote control. Ten GPS data points were taken and averaged when the robot was stationary to obtain an accurate initial position. Next, a waypoint was selected by clicking on the Google map interface. The waypoints were chosen approximately 40 meters away from the initial position so that the robot's performance of approaching a destination from far to close waypoints could be observed. This process was repeated 10 times.

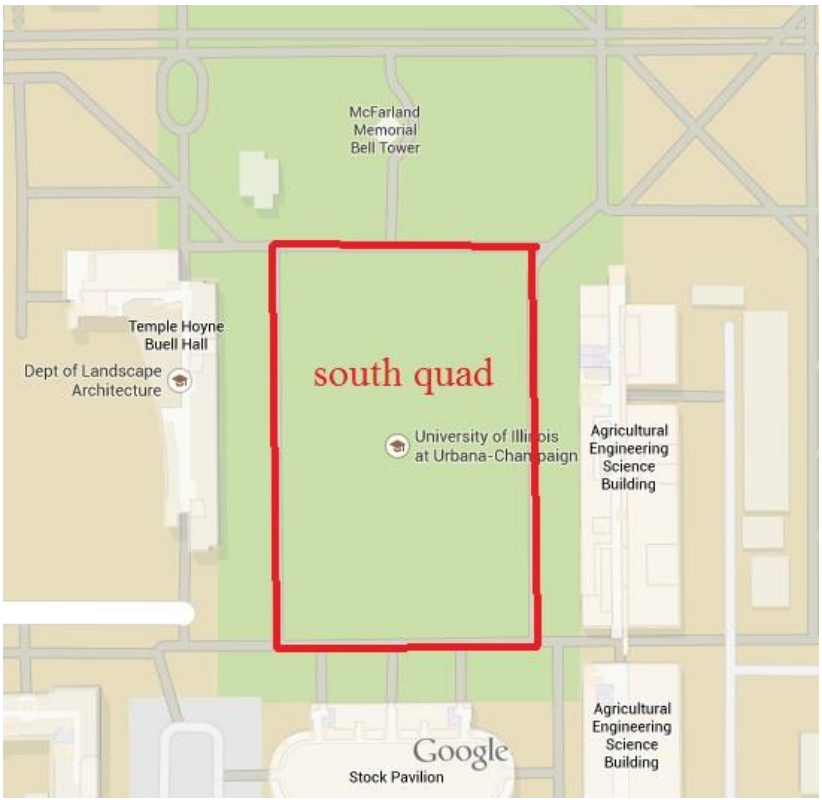

Figure 4.3 South quad in University of Illinois at Urbana-Champaign

4.2.3 Results and Discussion

The result for the experiments with solely GPS navigation showed that the robot never reached any waypoints and continued indefinitely. This was expected because the accuracy of the GPS was about 6 meters which was sufficient for guiding to robot toward a waypoint, but insufficient to reach it; When the distance between the robot and the waypoint was less than the GPS's accuracy (6m), the position of the robot relative to the waypoint became uncertain. Therefore, it would turn randomly due to the random error of GPS signals.

4.3 LIDAR Navigation

LIDAR Navigation was tested and results of the test were evaluated for comparison with GPS navigation and GPS/LIDAR navigation in section 4.5.

4.3.1 LIDAR Navigation Frame

The LIDAR navigation frame was a 2D coordinate system. The system was defined as follows: the location of the LIDAR was the origin, x axis was the direction parallel to the LIDAR's baseline and y axis was the direction vertical to the baseline (Figure 4.4). The range for x value was negative infinity to positive infinity and the range for y value was zero to positive infinity.

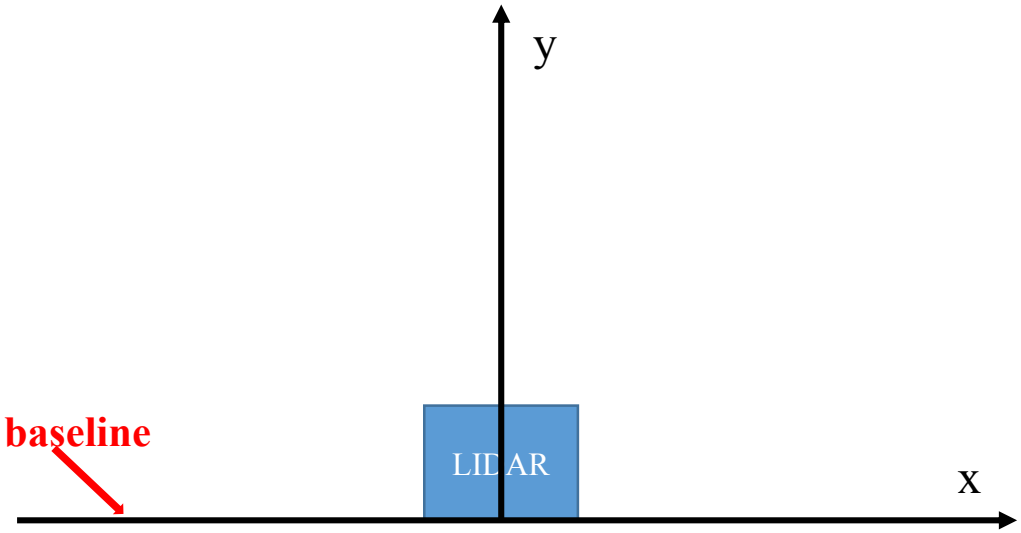

Figure 4.4 LIDAR's coordinate system

4.3.2 Data Processing

To guide the robot, the LIDAR navigation unit received data from the SICK LMS 291 unit and the PARALLAX electronic compass. The LIDAR provided location data in [x, y] pairs in centimeters in the LIDAR navigation frame and the compass transmitted heading information in degrees. The data transmitting frequency of the LIDAR was 1 Hz while the update frequency of the compass remained at 20 Hz. The information update strategy when calculating the robot's wheel speeds was the same as that for GPS navigation, being that the system would regard the location of the robot constant before the next LIDAR data was received and controlling the speed of the wheels solely based on the updated heading information.

4.3.3 Experiment Design

The Experiment Design for LIDAR navigation contained three factors: 1) selection of experiment location, 2) selection of LIDAR setting and 3) selection of waypoints. The test was also conducted on the south quad of University of Illinois at Urbana-Champaign. The LIDAR scanner was mounted at the south end center of the quad facing north. Note that there is no specific requirement of where the LIDAR scanner should be placed; the sole reason for the chosen location was that the ground surface was flat.

To detect the robot at all times, the LIDAR scanner needed to be at the same height as the robot. The robot was taken to the location of the LIDAR scanner under remote control. After all systems were checked and ready, the first step was to select a proper setting including range and mode of LIDAR from Table 3.2. Because the LIDAR aperture had to be 180 degrees, and the range was further than 8.2 meters, the range and mode selected were [0˚ - 180˚], 0.5˚ and cm Mode. This mode allows a detection range of 82 meters and, with an angular resolution of 0.5 ˚, had a potential localization accuracy better 0.65 meters with its range. After the range and mode were set up, the LIDAR was given 30 seconds to localize stationary objects such as trees, which were filtered. The last step was to select a waypoint. As in the GPS navigation test, the process was repeated 10 times and waypoints were selected in a range from approximately 30 to 70 meters from the initial position so that the robot's performance of approaching a destination from being far to being close could be observed.

4.3.4 Results and Discussion

The results are shown in two aspects: one is the display of the robot's trajectories which represent the robustness of LIDAR detection, and the other is the accuracy of LIDAR localization. Figure 4.5 and Figure 4.6 show two trajectories during the experiment. It can be seen that during

the navigation process, there were only two to three outliers, which were due to detection errors. Since the outliers were few, they did not affect the decision making process for controlling wheel speeds.

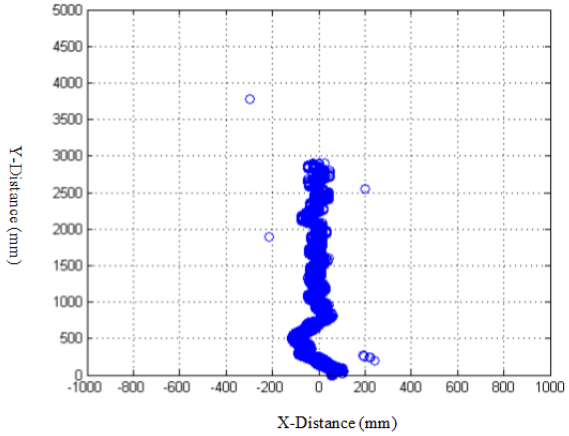

Figure 4.5 Robot's trajectory 1

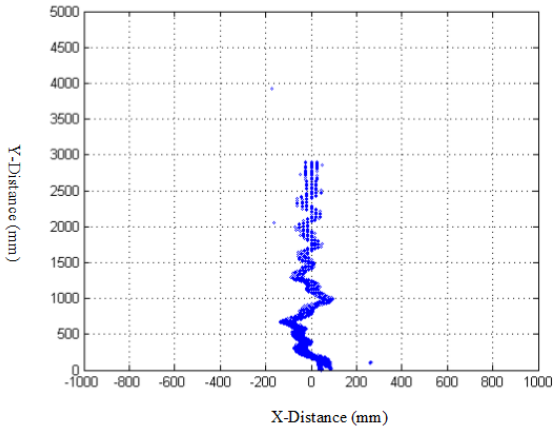

Figure 4.6 Robot's trajectory 2

The evaluation of accuracy of LIDAR navigation was to measure the distance between waypoints and robot's stop points. Statistical results for the ten runs are shown in Table 4.1 and Table 4.2. From the Table 4.2, it can be shown that the mean error was 0.194 m with a maximum error of 0.8 m. Note that, the maximum error occurred in test number 6, when the LIDAR detected outliers, so that the system estimated the position of the robot incorrectly. Figure 4.7 shows the LIDAR detected trajectory of test number 6 in which the waypoint was set at [0 mm, 2800 mm]. The small blue circle at position around [200 mm, 3500 mm] was detected at last moment of that run, and it was averaged with some other blue circles at position around [0 mm, 26 mm], which was the true location of the robot. The averaged location happened to be near to location [0 mm,

2800 mm], which made the system stop the robot. Considering that situation would rarely happen, and if the test number 6 is removed from table 4.1, the average error decreases to 0.127 m.

Table 4.1 LIDAR localization errors in experiment

Localization Error in 10 tests (m)									
1	2	3	4	5	6	7	8	9	10
0.07	0.13	0.25	0.05	0.2	0.8	0.06	0.12	0.09	0.17

Table 4.2 Statistics of errors of LIDAR localization

Average Error (m)	Average Error with outlier removed (m)	Max Error (m)	Min Error (m)	Error Standard Deviation
0.194	0.127	0.8	0.05	0.064

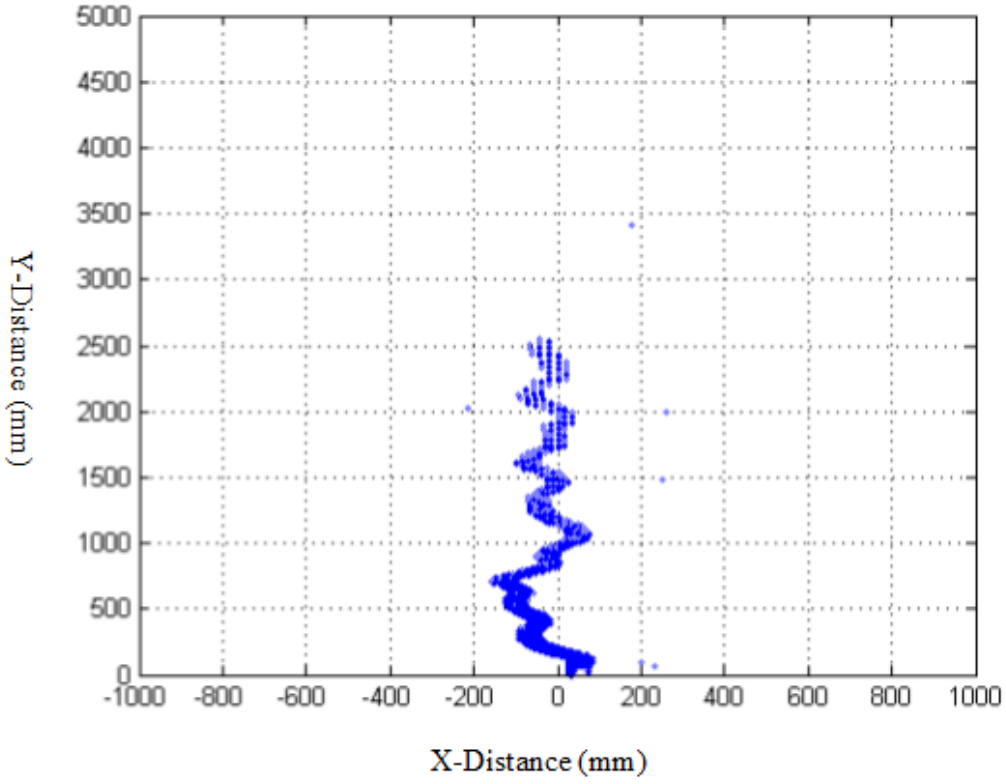

Figure 4.8 the robot's trajectory of test number 6

4.4 GPS and LIDAR Combined Navigation

The combined GPS/LIDAR navigation was tested at the same place with GPS navigation and LIDAR navigation and its results were evaluated for comparison with those two navigation systems in section 4.5.

4.4.1 Transformation between GPS navigation frame and LIDAR navigation frame

In the combined navigation system, the first step was to find the frame transformation matrix which would convert [Latitude, Longitude] pairs to [x, y] pairs. GPS data [Latitude, Longitude] pairs were taken in the earth frame in which the y axis points to the north and x axis points to the east. LIDAR data [x, y] pairs were taken in the self-defined navigation frame in which the y axis was set to point the north and x axis was set to point the east. Thus, the frame transformation matrix was simplified into a scale transformation matrix. The transformation matrix was defined as C_e^n. The state space equation is shown as the follows.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} F_{lon} 0 \\ 0 \quad F_{lat} \end{bmatrix} \left\{ \begin{bmatrix} Lat \\ Lon \end{bmatrix} - \begin{bmatrix} Lat0 \\ Lon0 \end{bmatrix} \right\}$$

$$F_{lon} = \frac{\pi \div 180 \times a^2 \times \cos(\text{Latitude(P 0)} \div 180 \times \pi)}{\sqrt{a^2 \times (\cos(\text{Lat} \div 180 \times \pi))^2 + b^2 \times (\sin(\text{Lat} \div 180 \times \pi))^2}}$$

$$F_{lat} = \frac{\pi \div 180 \times a^2 \times b^2}{\sqrt{a^2 \times (\cos(\text{Lat} \div 180 \times \pi))^2 + b^2 \times (\sin(\text{Lat} \div 180 \times \pi))^2}}$$

Where,

F_{lon} = distance corresponding to 1 degree change in latitude

F_{lat} = distance corresponding to 1 degree change in longitude

$a = 6378137m$

$b = 6356752.3142m$

$P0$ = position of the starting point

Lat = GPS latitude data of robot location

Lon = GPS longitude data of robot location

$Lat0$ = GPS latitude data of robot initial location

$Lon0$ = GPS longitude data of robot initial location

4.4.2 Experiment Design

The experiment design for the combined navigation was virtually identical to that of the LIDAR navigation experiment, because the LIDAR was the main information source for navigation. One difference was in the step of waypoints selection that multi-waypoints were selected in a single run to test the robustness of the control loop.

4.4.3 Results and Discussion

The results for the combined navigation included two parts: one is the display of robot trajectories and the other is the evaluation of navigation errors. Figure 4.7 and Figure 4.8 show the robot's trajectories with multiple-waypoints.

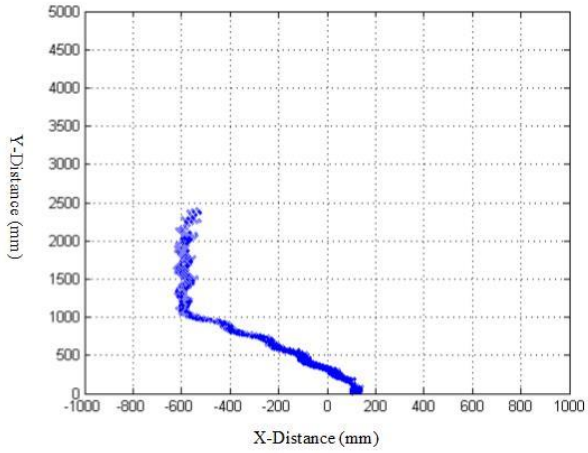

Figure 4.9 Robot's trajectory 3

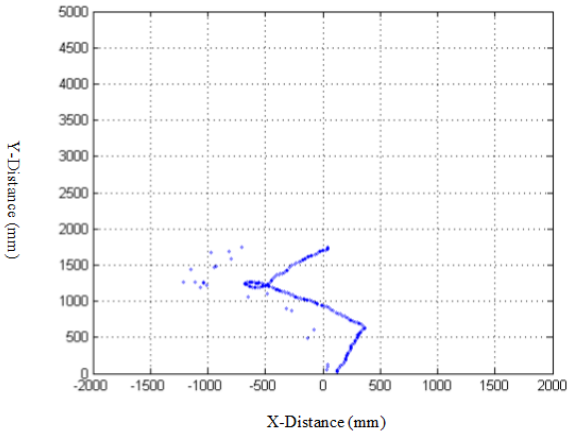

Figure 4.10 Robot's trajectory 4

Evaluation of the accuracy of the combined navigation is shown in Table 4.2. The mean error was 0.20 m with a maximum error of 0.33 m.

Table 4.3 Combined navigation error in experiment

Localization Error in 10 tests (m)									
1	2	3	4	5	6	7	8	9	10
0.25	0.20	0.13	0.07	0.33	0.15	0.25	0.28	0.17	0.19

Table 4.4 Statistics of Errors of combined navigation

Average Error (m)	Max Error (m)	Min Error (m)	Error Standard Deviation
0.20	0.33	0.07	0.077

4.5 Comparison among navigation systems

The results of three navigation systems were compared. The GPS navigation system offered localization information which could be used directly with Google maps. However, this method was not suitable for precise navigation, due to the low accuracy of GPS localization.

The LIDAR navigation system provided accurate positioning which allowed the robot to visit waypoints accurately with information of the positions of waypoints relative to the LIDAR scanner. The combined navigation had an accurate localization similar to LIDAR navigation. The accuracy was a little lower because the combined system had slower information update speed and the transformation between LIDAR frame and GPS navigation frame involved more errors. However, the combined navigation provided a convenient interface, where knowledge of relative position of waypoints to LIDAR was not required, while keeping the accuracy high. Additionally, the combined navigation compared the position information from GPS and LIDAR, and would exclude locations that did not match between these GPS and LIDAR. Thus, the chances of having outliers would decrease.

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS

The first part summarizes the work and achievement of this project, and addresses the objectives listed in section 1.2. The second part will present the insights and insufficiencies of this work.

5.1 Summary

A remote control network was built to manually control the robot. A Tower Hobbies System 3000 4 FM radio was selected as a remote control. A 2D control strategy was written that with its help the robot could move forward, move backward, turn forward left, turn forward right, turn backward left, turn backward right, perform left and right spin turns. The radio based remote control was very robust.

A GPS auto-navigation system was built for outdoor positioning. The GPS navigation system which consisted of a Garmin V GPS, a Parallax electronic compass as input hardware, could provide location and heading information for the robot. Its accuracy was about 6 meters relying on the accuracy of the GPS. The system could provide reliable information in outdoor situation when good weather was available.

A LIDAR auto-navigation system was developed for outdoor and indoor guidance. The LIDAR navigation system included a SICK LIDAR scanner and a Parallax electronic compass. It was capable to guide the robot in both indoor and outdoor conditions. Its accuracy of its positioning on the axis parallel to LIDAR'S baseline decreased linearly with an increasing range. However, the system could still achieve an accuracy of about 50 cm in the range of up to 80 meters.

The combination of GPS and LIDAR guidance provided more reliable navigation for the robot. The fused system made use of information from the GPS, the LIDAR and the compass for outdoor navigation. The accuracy of the system was evaluated based on experiments and it was about the same with the accuracy of LIDAR navigation system. Its advantage was that it was more robust than LIDAR navigation system because when working field was not perfectly even, LIDAR could temporarily lose the robot, in which case, the GPS could still offer position data. Both the LIDAR navigation and the combined navigation systems accomplished the goal of 1 m positioning accuracy set in section 1.2.

Google maps were used to offer easy user interface for setting waypoints. A local google map were plotted at the beginning of each navigation task and users could set waypoints by clicking on the map or typing in latitude and longitude pairs in known in advance. Thus, objective (3) listed in section 1.2 was accomplished.

The Xbee wireless communication network provided data transmission between the robot and the computer with up to 9600 baud rate. Transmitted position data was shown on the computer for users to monitor the positions and trajectories of the robot in real-time. Thus, both objective (2) and (4) were achieved.

5.2 Recommendations for Future Work

Regarding the limitations of the robot such as requiring flat work field, having short range, having low accuracy, and requiring good weather if using GPS, the usage of the robot will also be limited. More work can be done to make the robot more applicable.

5.2.1 Improving control algorithms

Algorithms can be developed to improve the positioning accuracy if accuracy of sensors is better studied. A Kalman Filter can be implemented to fuse GPS and LIDAR data if the covariance

matrixes of both sensors' errors can be estimated in real-time. This can be achieved by doing pre-experiment of GPS and LIDAR units. For example, the accuracy of LIDAR positioning can be measured with respect to different distances and the accuracy of GPS in good weather can easily be studied. With the predefined covariance matrixes of errors, Kalman Filter can probably give a much better estimation of the robot's position.

5.2.2 Adding Inertial sensors

Inertial sensors including accelerometers and gyroscopes can measure the robot's motion with a high frequency, which make them suitable for real time positioning if the initial position and heading is known. As mentioned in chapter 2, by adding inertial sensors, Kalman Filters will be used to fuse the inertial data and position data from either the GPS or the LIDAR. The limitation of requiring flat working field and having short range will be removed in good weather and the positioning accuracy could be improved dramatically.

5.2.3 Using vision-aided inertial navigation system

A robot with a vision-aided inertial navigation system has the potential to solve problems of all the limitations mentioned at the beginning of section 5.2. The basic control algorithm is still a Kalman Filter, where positioning errors from vision information are normally distributed. If the positioning errors are not normally distributed, a Bayesian filter can be used instead of Kalman filter to estimate the robot's status including positions, speeds, and attitudes. The advantages of this method are that the robot can work in any weather condition as long as working fields are visible. The shapes and levels of flatness will not be significant factors because in principle both sensors can work without well independent of the field conditions.

At last, there is no limitation of ranges when batteries' life and wireless communication are not concerns. This method, however, requires much more post data processing and may have

a limitation of processing speed of image data if algorithms are not developed properly or the computer's ability of processing data is not fast enough.

5.2.4 Using wifi network

As Xbee wireless communication usually has short ranges, it will be a major concern for limiting work ranges of the robot. A wifi communication does not have this limitation if both the control center which is the laptop in this project, and the robot are covered with wifi. This, of course, requires more investment to build a huge wifi network in the robot's working area, which may not realistic.

5.2.5 Software implementation

The major software used in this project was MATLAB, which is expensive for individual users and its processing speed is far from satisfactory if vision data are used. Possible solutions should have features such as requiring low-cost or being free to use and faster processing speeds. Potential candidates include Python and OpenCV. However, experiment need to be done to test if they are suitable for specific tasks.

REFERENCES

Bac, W., Grift, T., & Menezes, G. (2011). Development of a tabletop guidance system for educational robots. *Applied Engineering in Agriculture*, 27(5): 829-838.

Borenstein, J., Everett, H. R., Feng, L., & Wehe, D. (1997). Mobile robot positioning-sensors and techniques. *AVAL COMMAND CONTROL AND OCEAN SURVEILLANCE CENTER RDT AND E DIV SAN DIEGO CA*.

Carter, M. (1993). *Soil sampling and methods of analysis.* CRC Press.

Cole, D. M., & Newman, P. M. (2006). Using laser range data for 3D SLAM in outdoor environments. *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference*.

Diosi, A., & Kleeman, L. (2004). Advanced sonar and laser range finder fusion for simultaneous localization and mapping. *ntelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference.*

Farrell, J. (2008). *Aided navigation: GPS with high rate sensors.* New York: McHill.

Feuerbacher, B., & Stoewer, H. (2006). *Utilization of Space: today and tomorrow*. Springer.

Fu, S., Liu, H. Y., Gao, L., & Gai, Y. X. (2007). LAM for mobile robots using laser range finder and monocular vision. *Mechatronics and Machine Vision in Practice, 2007. M2VIP 2007. 14th International Conference.*

Geier, G. J. (1996). Odometer assisted GPS navigation method. *U.S. Patent No. 5,525,998. Washington, DC: U.S. Patent and Trademark Office.*

Grift, T. E., & Kasten, M. N. (2005). Robotics in agriculture: Asimov meets corn. *In: Proc. 2005 Illinois Crop Protection Technology Conference.* Urbana, IL.

Grift, T., Kondo, N., & Ting, K. (2008). Review of automation and robotics for the bio-industry. *Journal of Biomechatronics Engineering*, 1(1):37-54.

Gutmann, J. S., Weigel, T., & Nebel, B. (2001). A fast, accurate and robust method for self-localization in polygonal environments using laser range finders. *Advanced Robotics*, 651-667.

Kurazume, R., & Hirose, S. (1998). Study on cooperative positioning system: optimum moving strategies for CPS-III. *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference.*

Núñez, P., Vázquez-Martín, R., Del Toro, J. C., Bandera, A., & Sandoval, F. (2008). Natural landmark extraction for mobile robot navigation based on an adaptive curvature estimation. *obotics and Autonomous Systems*, 247-264.

Ohno, K., Tsubouchi, T., Shigematsu, B., & Yuta, S. I. (2004). Differential GPS and odometry-based outdoor navigation of a mobile robot. *Advanced Robotics*, 611-635.

Piersimoni, F., & Bee, M. (2010). *Agricultural survey methods.*

Qi, H., & Moore, J. B. (2002). Direct Kalman filtering approach for GPS/INS integration. *Aerospace and Electronic Systems, IEEE Transactions on 38.2* , 687-693.

Raible, J., Blaich, M., & Bittel, O. (2010). Differential GPS supported navigation for a mobile robot. *Intelligent Autonomous Vehicles, 7*, 318-323.

Rencken, W. D. (1993). Concurrent localisation and map building for mobile robots using ultrasonic sensors. *Intelligent Robots and Systems' 93, IROS'93. Proceedings of the 1993 IEEE/RSJ International Conference.*

Richard, B. F., & Gary, W. H. (1994). Soil Sampling for Precision Agriculture.

Roumeliotis, S. I., & Bekey, G. A. (2000). Segments: A layered, dual-kalman filter algorithm for indoor feature extraction. *Proceedings. 2000 IEEE/RSJ International Conference* .

Sanchez, P. A., Ahamed, S., Carré, F., Hartemink, A. E., Hempel, J., Huising, J., & Lagacherie, P. (2009). Digital soil map of the world. *Science*, 325.

Sasiadek, J. Z., & Wang, Q. (1999). Sensor fusion based on fuzzy Kalman filtering for autonomous robot vehicle. *In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, 2970-2975.

Sukkarieh, S., Nebot, E. M., & Durrant-Whyte, H. F. (1999). A high integrity IMU/GPS navigation loop for autonomous land vehicle applications. *Robotics and Automation, IEEE Transactions on 15(3)*, 572-578.

Surmann, H., Lingemann, K., Nüchter, A., & Hertzberg, J. (2001). A 3D laser range finder for autonomous mobile robots. *roceedings of the 32nd ISR (International Symposium on Robotics).*

Titterton, D., & Weston, J. (2004). *Strapdown Inertial Navigation Technology.* IET.

Xue, J. L., Zhang, & Grift, T. (2012). Variable field of view-machine vision based row guidance of an agricultural robot. *Computers and Electronics in Agriculture*, 84: 85-91.

Xue, J., & Grift, T. (2011). Agricultural robot turning in the headland of corn fields. *Applied Mechanics and Materials*, 63-64, 780-784.

Zhang, P., Gu, J., Milios, E. E., & Huynh, P. (2005). Navigation with IMU/GPS/Digital Compass with. *Mechatronics and Automation, 2005 IEEE International Conference*, 1497-1502.

Zhang, S., Xie, L., & Adams, M. D. (n.d.). Feature extraction for outdoor mobile robot navigation based on a modified Gauss–Newton optimization approach. *obotics and Autonomous Systems*, 277-287.

APPENDIX A: ARDUINO IMPLEMENTATION CODE

```
   #include <Wire.h>
   #include <math.h>
   //#include <SoftwareSerial.h>
   //-------------------------------------------------------
   // read GPS data setting
   //#define rxPin 2
   //#define txPin 3   // GPS serial communication ports
   //SoftwareSerial gps = SoftwareSerial(rxPin,txPin);
   int  index1=0;
   char GPSString[56];
   char NonString[4];
   //-------------------------------------------------------
   #define Addr 0x1E                  // 7-bit address of HMC5883
compass
   int          compass_start = 1;
   int          remotePin    = A0; //set the input pin for remote
control switch
   int          RC        = 0;
   int          remote    = 0;

   const int    LwheelPin = 10;
   const int    RwheelPin = 11;
   const int    DIS_r     = 8;
   const int    S_r       = 9;
   const int    S_l       = 12;
   const int    DIS_l     = 13;

   word   speed_ll;
   word   speed_rr;
   int    speed_l =0;
   int    speed_r =0;
   word   THRO_puls;
   word   RUDD_puls;
   word   THRO_speed;
   word   RUDD_speed;
   byte   back_factor;

   byte   rob_drive_mod;
   // 0  =  stop,  spinturn  left  or  spinturn  right
   // 1  =  forwards
   // 2  =  backwards
```

```
byte drive_modus;
//     1 = forwards
//     2  =  backwards
//  3  =  spinturn  left
//  4  =  spinturn  right

//-----Preferences RC-----
const int    MIN_RUDD     = 1000;
const int    MIN_THRO     = 1180;
const int    MIDDLE_RUDD  = 1470;
const int    MIDDLE_THRO  = 1540;
const int    MAX_RUDD     = 1850;
const int    MAX_THRO     = 1910;
const int    deadband     = 50;
//--------------------------
// Pins 22 and 24 are available for receiving signal
const int    RUDD         = 22;
const int    THRO         = 24;
//--------------------------
void setup(){

  pinMode(DIS_r,OUTPUT);
  pinMode(S_r,OUTPUT);
  pinMode(S_l,OUTPUT);
  pinMode(DIS_l,OUTPUT);

  pinMode(RUDD,INPUT);
  pinMode(THRO,INPUT);

  Serial.begin(9600);

  // compass set up
  Wire.begin();
  // Set operating mode to continuous
  Wire.beginTransmission(Addr);
  Wire.write(byte(0x02));
  Wire.write(byte(0x00));
  Wire.endTransmission();

  // Read GPS set up
  //pinMode(rxPin, INPUT);
  //pinMode(txPin, OUTPUT);
  //gps.begin(4800);
  Serial3.begin(4800);
}
```

```
  void loop(){
    remote_control();
    if (remote ==1)
    {
      check_channels();
      drive();
    }
    else
    {
      //d_modus_1();

      gps_control();

    }
  }

  void remote_control(){
    RC   =   analogRead(remotePin);
    if(RC > 50)
    {
      remote = 1;
    }
    else
    {
      remote  =0;
    }
  }

  void stop_robot(){
    analogWrite(LwheelPin,0);
    analogWrite(RwheelPin,0);
    loop();
  }

  void check_channels(){
    THRO_puls  = pulseIn(THRO,HIGH);
    RUDD_puls  = pulseIn(RUDD,HIGH);

    if   ((THRO_puls   >   MAX_THRO   ||   THRO_puls   <   MIN_THRO)
||(RUDD_puls > MAX_RUDD || RUDD_puls < MIN_RUDD)){
      rob_drive_mod = 0;
      stop_robot();
    }

    if ((THRO_puls > (MIDDLE_THRO - deadband)) && (THRO_puls <
(MIDDLE_THRO + deadband)))
    {
```

```
      rob_drive_mod = 0;
      speed_r       = 0;
      speed_l       = 0;
    }
    else if(THRO_puls < (MIDDLE_THRO - deadband))
    {
      rob_drive_mod = 1;
      THRO_speed = (MIDDLE_THRO - THRO_puls) * 11/2;
      if (THRO_speed > 2048) {
        THRO_speed = 2000;
      }
    }
    else if(THRO_puls > (MIDDLE_THRO + deadband))
    {
      rob_drive_mod = 2;
      THRO_speed = (THRO_puls - MIDDLE_THRO) * 11/2;
      if (THRO_speed > 2048){
        THRO_speed = 2000;
      }
    }

  }

  void drive(){
    if (rob_drive_mod == 0){
      if ((RUDD_puls >= (MIDDLE_RUDD - deadband)) && (RUDD_puls
<= (MIDDLE_RUDD + deadband)))
      {
        stop_robot();
      }
      else if(RUDD_puls < (MIDDLE_RUDD - deadband))
      {
        RUDD_speed = (MIDDLE_RUDD - RUDD_puls) * 3;
        spinturn_left();
      }
      else if(RUDD_puls > (MIDDLE_RUDD + deadband))
      {
        RUDD_speed = (RUDD_puls - MIDDLE_RUDD) * 3;
        spinturn_right();
      }
    }

    if(rob_drive_mod == 1){
      if(drive_modus != 1){
        d_modus_1();
      }
```

```
      if((RUDD_puls >= (MIDDLE_RUDD - deadband)) && (RUDD_puls <=
(MIDDLE_RUDD + deadband)))
      {
        speed_l = THRO_speed;
        speed_r = THRO_speed;
      }
      else if(RUDD_puls > (MIDDLE_RUDD + deadband))
      {
        speed_l = THRO_speed;
        RUDD_speed = THRO_speed/2;
        speed_r =  RUDD_speed;
      }
      else if(RUDD_puls < (MIDDLE_RUDD - deadband))
      {
        speed_r = THRO_speed;
        RUDD_speed = THRO_speed/2;
        speed_l = RUDD_speed;
      }
    }

    if(rob_drive_mod == 2){
      if (drive_modus != 2)
      {
        d_modus_2();
      }
      if((RUDD_puls >= (MIDDLE_RUDD - deadband)) && (RUDD_puls <=
(MIDDLE_RUDD + deadband)))
      {
        speed_l = THRO_speed;
        speed_r = THRO_speed;
      }
      else if(RUDD_puls > (MIDDLE_RUDD + deadband))
      {
        speed_l = THRO_speed;
        RUDD_speed = THRO_speed/2;
        speed_r =  RUDD_speed;
      }
      else if(RUDD_puls < (MIDDLE_RUDD - deadband))
      {
        speed_r = THRO_speed;
        RUDD_speed = THRO_speed/2;
        speed_l = RUDD_speed;
      }
    }
    speed_ll = map(speed_l,0,2047,0,255);
    speed_rr = map(speed_r,0,2047,0,255);
    analogWrite(LwheelPin, speed_ll);
```

```
  analogWrite(RwheelPin, speed_rr);

}

void spinturn_left(){
  if (drive_modus != 3){
    d_modus_3();
  }
  if (RUDD_speed >= 1500)
  {
    speed_l = 1500;
    speed_r = 1500;
  }
  else
  {
    speed_l = RUDD_speed;
    speed_r = RUDD_speed;
  }
  speed_ll = map(speed_l,0,2047,0,255);
  speed_rr = map(speed_r,0,2047,0,255);
  analogWrite(LwheelPin, speed_ll);
  analogWrite(RwheelPin, speed_rr);

}

void spinturn_right(){
  if (drive_modus != 4){
    d_modus_4();
  }
  if (RUDD_speed >= 1500)
  {
    speed_l = 1500;
    speed_r = 1500;
  }
  else
  {
    speed_l = RUDD_speed;
    speed_r = RUDD_speed;
  }
  speed_ll = map(speed_l,0,2047,0,255);
  speed_rr = map(speed_r,0,2047,0,255);
  analogWrite(LwheelPin, speed_ll);
  analogWrite(RwheelPin, speed_rr);

}

void d_modus_1(){
```

```
  digitalWrite(DIS_l,HIGH);
  digitalWrite(DIS_r,HIGH);
  delay(1);
  digitalWrite(S_r,LOW);
  digitalWrite(S_l,HIGH);
  delay(1);
  digitalWrite(DIS_l,LOW);
  digitalWrite(DIS_r,LOW);
  drive_modus = 1;

}

void d_modus_2(){
  digitalWrite(DIS_l,HIGH);
  digitalWrite(DIS_r,HIGH);
  delay(1);
  digitalWrite(S_r,HIGH);
  digitalWrite(S_l,LOW);
  delay(1);
  digitalWrite(DIS_l,LOW);
  digitalWrite(DIS_r,LOW);
  drive_modus = 2;

}

void d_modus_3(){
  digitalWrite(DIS_l,HIGH);
  digitalWrite(DIS_r,HIGH);
  delay(1);
  digitalWrite(S_r,HIGH);
  digitalWrite(S_l,HIGH);
  delay(1);
  digitalWrite(DIS_l,LOW);
  digitalWrite(DIS_r,LOW);
  drive_modus = 3;

}

void d_modus_4(){
  digitalWrite(DIS_l,HIGH);
  digitalWrite(DIS_r,HIGH);
  delay(1);
  digitalWrite(S_r,LOW);
  digitalWrite(S_l,LOW);
  delay(1);
  digitalWrite(DIS_l,LOW);
  digitalWrite(DIS_r,LOW);
```

```
    drive_modus = 4;

  }

  void gps_control(){
    //read_gps_data();
    char* StrGPS=read_gps_data();
    if
(StrGPS[0]=='$'&&StrGPS[1]=='G'&&StrGPS[2]=='P'&&StrGPS[3]=='R'&
&StrGPS[4]=='M'&&StrGPS[5]=='C'){
    char* StrHeading=compass_matlab();

    char Sendoutstring[100];
    Sendoutstring[0]= 0;
    strcat(Sendoutstring,StrGPS);
    strcat(Sendoutstring,StrHeading);
    Serial.println(Sendoutstring);
    }
    else                                                    if
(StrGPS[0]=='N'&&StrGPS[1]=='o'&&StrGPS[2]=='n'&&StrGPS[3]=='e')
{
     char* StrHeading=compass_matlab();
     char Sendoutstring[100];
     Sendoutstring[0]=0;
     strcat(Sendoutstring,StrGPS);
     strcat(Sendoutstring,StrHeading);
     Serial.println(Sendoutstring);
     delay(250);
    }

    if(Serial.available() > 0)
    {
      delay(5);
      char incomdirec = Serial.read();


        int intergerValue = 0;
             while(1){
             char incomint = Serial.read();
             if (incomint == '\n') break;
             intergerValue *= 10;
             intergerValue = ((incomint - 48) + intergerValue);
             delay(5);
             }
        speed_l = intergerValue;
```

```
   intergerValue = 0;
       while(1){
       char incomint = Serial.read();
       if (incomint == '\n') break;
       intergerValue *= 10;
       intergerValue = ((incomint - 48) + intergerValue);
       delay(5);
       }
  speed_r = intergerValue;

  Serial.println(speed_l,DEC);
  Serial.println(speed_r,DEC);

if(incomdirec=='a')
{
  digitalWrite(DIS_l,HIGH);
  digitalWrite(DIS_r,HIGH);
  delay(1);
  digitalWrite(S_r,LOW);
  digitalWrite(S_l,HIGH);
  delay(1);
  digitalWrite(DIS_l,LOW);
  digitalWrite(DIS_r,LOW);
  analogWrite(LwheelPin, speed_l);
  analogWrite(RwheelPin, speed_r);
}
  else if(incomdirec=='b')
  {
    digitalWrite(DIS_l,HIGH);
    digitalWrite(DIS_r,HIGH);
    delay(1);
    digitalWrite(S_r,LOW);
    digitalWrite(S_l,LOW);
    delay(1);
    digitalWrite(DIS_l,LOW);
    digitalWrite(DIS_r,LOW);
    speed_l = speed_l-180;
    speed_r = speed_r-180;
    analogWrite(LwheelPin, speed_l);
    analogWrite(RwheelPin, speed_r);
  }
  else if (incomdirec=='c')
  {
    digitalWrite(DIS_l,HIGH);
    digitalWrite(DIS_r,HIGH);
```

```
          delay(1);
          digitalWrite(S_r,HIGH);
          digitalWrite(S_l,HIGH);
          delay(1);
          digitalWrite(DIS_l,LOW);
          digitalWrite(DIS_r,LOW);
          speed_l = speed_l-180;
          speed_r = speed_r-180;
          analogWrite(LwheelPin, speed_l);
          analogWrite(RwheelPin, speed_r);
        }
        Serial.println(incomdirec);
    }
    }

  char* compass_matlab() {
    int x, y, z;
    float theta;
    float heading;
    // Initiate communications with compass
    Wire.beginTransmission(Addr);
    Wire.write(byte(0x03));      // Send request to X MSB register
    Wire.endTransmission();

    Wire.requestFrom(Addr, 6);    // Request 6 bytes; 2 bytes per
axis
    if(Wire.available() <=6) {    // If 6 bytes available

      x = Wire.read() << 8 | Wire.read();
      z = Wire.read() << 8 | Wire.read();
      y = Wire.read() << 8 | Wire.read();
      // Calculate heading when the magnetometer is level, then
correct for signs of axis.
      heading = atan2(y,x);

      // Your mrad result / 1000.00 (to turn it into radians).
      float declinationAngle = 3.02/180*PI;
      // If you have an EAST declination, use += declinationAngle,
if you have a WEST declination, use -= declinationAngle
      heading += declinationAngle;

      // Correct for when signs are reversed.
      if(heading < 0)
        heading += 2*PI;

      // Check for wrap due to addition of declination.
      if(heading > 2*PI)
```

```
        heading -= 2*PI;

      // Convert radians to degrees for readability.
      float headingDegrees = heading * 180/M_PI;
      char headingstring[10];
      dtostrf(headingDegrees,2,2,headingstring);


      return headingstring;
      delay(10);
    }
  }

  char* read_gps_data(){

    char incomingByte;

    if (Serial3.available()>0){     //capture NMEA sentence and
print to serial window
      delay(5);

      if(Serial3.read()=='$'){

        GPSString[0]='$';
        delay(5);
        for (index1 = 1; index1 < 55; index1 = index1 + 1) {
          if (Serial3.available()){
            delay(7);
        incomingByte = Serial3.read();
        GPSString[index1] = incomingByte;
          }

        }
        GPSString[55] = '|';


        if
(GPSString[0]=='$'&&GPSString[1]=='G'&&GPSString[2]=='P'&&GPSStr
ing[3]=='R'&&GPSString[4]=='M'&&GPSString[5]=='C'){
          return GPSString;
        }
        else{
          NonString[0]='N';
          NonString[1]='o';
          NonString[2]='n';
          NonString[3]='e';
          return NonString;
```

```
        }
       delay(100);
     }
     else{
       NonString[0]='N';
       NonString[1]='o';
       NonString[2]='n';
       NonString[3]='e';
       return NonString;
     }
   }
    else{
      NonString[0]='N';
      NonString[1]='o';
      NonString[2]='n';
      NonString[3]='e';
      return NonString;
    }
}
```

APPENDIX B: MATLAB IMPLEMENTATION CODE

LIDAR_GPS Control.m

```
clear all;
global Serial_Port;
global Navi;
Kv              = 0.3;
Kh              = 0.5;
% XWay           = 0;
% YWay           = 3000;
X_collect       = [];
Y_collect       = [];
Waypoint        = 3;
% Plot google map and generate waypoints
Latset   =   [40.102493998448665            40.101406639707236
40.101414846253384    40.10250630807077];
Lonset  =  [-88.22759091854095       -88.22758555412292       -
88.22667360305786    -88.22667360305786];
plot(Lonset,Latset,'.r','MarkerSize',10);
plot_google_map;
[Lon_Way, Lat_Way]=ginput(Waypoint);

% Initialize GPS to get accurate position
Navi           =           serial('COM13',           'BaudRate',
9600,'InputBufferSize',2048);
Navi.ReadAsyncMode = 'manual';
[Lon0,Lat0]= GPS_Initialize;

% Initialize LIDAR to remove all stationary objects
[X1,Y1]=LIDAR_Initialize;
% Start robot--move it to the area where LIDAR can see

fopen(Navi);
fprintf(Navi,'%s', 'a');
fprintf(Navi,'%d\n',250);
fprintf(Navi,'%d\n',250);
pause(2);
fprintf(Navi,'%s','a');
fprintf(Navi,'%d\n',0);
fprintf(Navi,'%d\n',0);
pause(0.01);
fclose(Navi);
```

```
   disp('GOOOOOOOO');
   pause(2);
   % Convert [Lat0,Lon0] to LIDAR coordinate [X0,Y0]
   [X0,Y0] = ConvertGPS2LIDAR(X1,Y1);
    fopen(Navi);
   for i= 1:Waypoint
       X_Old          = 0;
       Y_Old          = 0;
       DistOld        = 10000;
       Lspeed         = 0;
       Rspeed         = 0;
       Dist           = 10000;
       HeadWayOld     = 0;
       HeadingOld     = 0;
       LonWay         = Lon_Way(i);
       LatWay         = Lat_Way(i);
       % Convert [LonWay,LatWay] to LIDAR coordinate[Xway,Yway]
       [XWay,YWay] = GPS2LIDAR(LonWay,LatWay,Lon0,Lat0,X0,Y0);
       %-----------------------------------------------

       % start control
       while(Dist>100)
           % Read GPS data
           [Lon,Lat,Heading]=readGPS_xbee_new4(Lspeed,Rspeed);
           if isempty(Heading)
               Heading = HeadingOld;
           end
           HeadingOld = Heading;
           % Convert [Lon,Lat] to LIDAR coordinate [Xway,Yway]
           [X_gps,Y_gps] = GPS2LIDAR(Lon,Lat,Lon0,Lat0,X0,Y0);
           % Get LIDAR [X,Y]
           [X_lidar,Y_lidar]= LIDAR_read(X1,Y1);
           Dist = 1000;
           [X_true,Y_true]=
Data_analysis(X_gps,Y_gps,X_lidar,Y_lidar,X_Old,Y_Old);
           X_collect = [X_collect X_true];
           Y_collect = [Y_collect Y_true];
           XNew  = X_true;
           YNew  = Y_true;
           X_Old = X_true;
           Y_Old = Y_true;
           Dist = sqrt((XNew-XWay)^2+(YNew-YWay)^2);
           % Calculate HeadWay-----------------------------------
--
           if XWay == XNew
               if YWay < YNew
                   HeadWay = 180;
```

```
            elseif YWay > YNew
                HeadWay = 0;
            else
                HeadWay = 0;
            end
        end

        if YNew == YWay
            if XNew < XWay          %Moving West
                HeadWay = 90;
            elseif XNew > XWay
                HeadWay = 270;            % Moving East
            else
                HeadWay = 0;            % Stationary
            end
        end
        % 1st quadrant
        if ((XWay> XNew) & (YWay > YNew))
            HeadWay = 180/pi*tanh(abs(XNew - XWay) / abs(YWay -
YNew));
        end

        % 2nd quadrant
        if ((XWay > XNew) & (YWay < YNew))
            HeadWay = 90 + 180/pi*tanh(abs(YNew - YWay) /
abs(XNew - XWay));
        end

        % 3rd quadrant
        if ((XWay < XNew) & (YWay < YNew))
            HeadWay = 180 + 180/pi*tanh(abs(XWay - XNew) /
abs(YNew - YWay));
        end

        % 4th quadrant
        if ((XWay < XNew) & (YWay > YNew))
            HeadWay = 270 + 180/pi*tanh(abs(YWay - YNew) /
abs(XWay - XNew));
        end
        %-------------------------------------------------------
-----------------------------------------------

        Velocity = Kv*Dist;
        if Velocity > 150
            Velocity = 150;
        end
        R        = Kh*(HeadWay - Heading);
```

```
            if HeadWay>60 & HeadWay<180
                if (Heading<HeadWay+60) & (Heading>HeadWay-60)
                    Lspeed = Velocity+R;
                    Rspeed = Velocity-R;
                elseif (Heading>=HeadWay+60)&(Heading<HeadWay+180)
                    Lspeed = -250;
                    Rspeed = 250;
                else
                    Lspeed = 250;
                    Rspeed = -250;
                end

            elseif (HeadWay>=180)&(HeadWay<300)

                if (Heading<HeadWay+60) & (Heading>HeadWay-60)
                    Lspeed = Velocity+R;
                    Rspeed = Velocity-R;
                elseif (Heading<=HeadWay-60)&(Heading>HeadWay-180)
                    Lspeed = 250;
                    Rspeed = -250;
                else
                    Lspeed = -250;
                    Rspeed = 250;
                end
            elseif (HeadWay<=60)&(HeadWay>0)
                if (Heading<=HeadWay+60)
                    Lspeed = Velocity +R;
                    Rspeed = Velocity -R;
                elseif (Heading>HeadWay+60)&(Heading<HeadWay+180)
                    Lspeed = -250;
                    Rspeed = 250;
                elseif
(Heading>=HeadWay+180)&(Heading<HeadWay+300)
                    Lspeed = 250;
                    Rspeed = -250;
                else
                    Lspeed = Velocity+Kh*(360+HeadWay - Heading);
                    Rspeed = Velocity-Kh*(360+HeadWay - Heading);
                end

            else
                if (Heading>=HeadWay-60)
                    Lspeed = Velocity +R;
                    Rspeed = Velocity -R;
                elseif (Heading<HeadWay-60)&(Heading>HeadWay-180)
                    Lspeed= 250;
```

```
                Rspeed= -250;
            elseif     (Heading<=HeadWay-180)&(Heading>HeadWay-
300)
                Lspeed = -250;
                Rspeed = 250;
            else
                Lspeed = Velocity+Kh*(HeadWay - Heading-360);
                Rspeed = Velocity-Kh*(HeadWay - Heading-360);
            end
        end




        if Lspeed > 250
            Lspeed = 250;
        elseif Lspeed < -250
            Lspeed = -250;
        end
        if Rspeed > 250
            Rspeed = 250;
        elseif Rspeed < -250
            Rspeed = -250;
        end
        if abs(Lspeed)~=250 & abs(Rspeed)~=250
            if Lspeed>=Rspeed
                Lspeed = 250;
                Rspeed = Rspeed+(250-Lspeed);
            else
                Rspeed = 250;
                Lspeed = Lspeed+(250-Rspeed);
            end
        end

        Lspeed = ceil(Lspeed);
        Rspeed = ceil(Rspeed);
        pause(0.01);
    end
    fprintf(Navi,'%s', 'a');
    fprintf(Navi,'%d\n',0);
    fprintf(Navi,'%d\n',0);
    Waypoint_task(Lon,Lat,X_gps,Y_gps);
end
plot(X_collect,Y_collect,'o','MarkerSize',2);
axis([-2000,2000,0,5000]);
grid on;
fclose(Navi);
```

```
   fclose(Serial_Port);
   delete(Serial_Port);
   delete(Navi);
```

plot_google map.m

```
   function varargout = plot_google_map(varargin)
   % function h = plot_google_map(varargin)
   % Plots a google map on the current axes using the Google Static
Maps API
   %
   % USAGE:
   % h = plot_google_map(Property, Value,...)
   % Plots the map on the given axes. Used also if no output is
specified
   %
   % Or:
   % [lonVec latVec imag] = plot_google_map(Property, Value,...)
   % Returns the map without plotting it
   %
   % PROPERTIES:
   %    Height (640)   - Height of the image in pixels (max 640)
   %    Width  (640)   - Width of the image in pixels (max 640)
   %     Scale (2)       - (1/2) Resolution scale factor . using
Scale=2 will
   %                      double the resulotion of the downloaded
image (up
   %                       to 1280x1280) and will result in finer
rendering,
   %                     but processing time will be longer.
   %    MapType        - ('roadmap') Type of map to return. Any of
[roadmap,
   %                     satellite, terrain, hybrid] See the Google
Maps API for
   %                     more information.
   %    Alpha (1)      - (0-1) Transparency level of the map (0 is
fully
   %                     transparent). While the map is always
   %                     moved to the bottom of the plot (i.e. will
   %                        not hide previously drawn items), this
can
   %                     be useful in order to increase readability
   %                      if many colors are ploted (using SCATTER
   %                      for example).
   %       ShowLabels  (1)  -  (0/1)  Controls  wheter  to  display
city/street textual labels on the map
```

```
   %    Marker          - The marker argument is a text string with
fields
   %                      conforming to the Google Maps API. The
   %                      following are valid examples:
   %                       '43.0738740,-70.713993' (default midsize
orange marker)
   %                     '43.0738740,-70.713993,blue' (midsize blue
marker)
   %                      '43.0738740,-70.713993,yellowa' (midsize
yellow
   %                      marker with label "A")
   %                        '43.0738740,-70.713993,tinyredb' (tiny
red marker
   %                      with label "B")
   %     Refresh (1)      - (0/1) defines whether to automatically
refresh the
   %                      map upon zoom/pan action on the figure.
   %     AutoAxis (1)     - (0/1) defines whether to automatically
adjust the axis
   %                           of the plot to avoid the map being
stretched.
   %                      This will adjust the span to be correct
   %                      according to the shape of the map axes.
   %     APIKey           - (string) set your own API key which you
obtained from Google:
   %
http://developers.google.com/maps/documentation/staticmaps/#api_
key
   %                     This will enable up to 25,000 map requests
per day,
   %                     compared to a few hundred requests without
a key.
   %                      To set the key, use:
   %
plot_google_map('APIKey','SomeLongStringObtaindFromGoogle')
   %                       You need to do this only once to set the
key.
   %                      To disable the use of a key, use:
   %                      plot_google_map('APIKey','')
   %
   % OUTPUT:
   %    h               - Handle to the plotted map
   %
   %    lonVect          - Vector of Longidute coordinates (WGS84)
of the image
   %    latVect         - Vector of Latidute coordinates (WGS84) of
the image
```

```
%    imag           - Image matrix (height,width,3) of the map
%
% EXAMPLE - plot a map showing some capitals in Europe:
%     lat = [48.8708    51.5188    41.9260    40.4312    52.523
37.982];
%    lon = [2.4131    -0.1300    12.4951    -3.6788    13.415
23.715];
%    plot(lon,lat,'.r','MarkerSize',20)
%    plot_google_map
%
% References:
% http://www.mathworks.com/matlabcentral/fileexchange/24113
%  http://www.maptiler.org/google-maps-coordinates-tile-bounds-
projection/
% http://developers.google.com/maps/documentation/staticmaps/
%
%    Acknowledgement to Val Schmidt for his submission of
get_google_map.m
%
%  Author:
%  Zohar Bar-Yehuda
% Version 1.3 - 06/10/2013
%       - Improved functionality of AutoAxis, which now handles
any shape of map axes.
%         Now also updates the extent of the map if the figure
is resized.
%        - Added the ShowLabels param which allows hiding the
textual labels on the map.
% Version 1.2 - 16/06/2012
%       - Support use of the "scale=2" parameter by default for
finer rendering (set scale=1 if too slow).
%       - Auto-adjust axis extent so the map isn't stretched.
%        - Set and use an API key which enables a much higher
usage volume per day.
%  Version 1.1 - 25/08/2011

% store parameters in global variable (used for auto-refresh)
global inputParams
persistent apiKey
if isnumeric(apiKey)
    % first run, check if API key file exists
    if exist('api_key.mat','file')
        load api_key
    else
        apiKey = '';
    end
end
```

```
    axHandle = gca;
    inputParams.(['ax' num2str(axHandle*1e6,'%.0f')]) = varargin;

    % Handle input arguments

    height = 640;
    width = 640;
    scale = 2;
    maptype = 'roadmap';
    alphaData = 1;
    autoRferesh = 1;
    autoAxis = 1;
    ShowLabels = 1;
    hold on

    markeridx = 1;
    markerlist = {};
    if nargin >= 2
        for idx = 1:2:length(varargin)
            switch lower(varargin{idx})
                case 'height'
                    height = varargin{idx+1};
                case 'width'
                    width = varargin{idx+1};
                case 'maptype'
                    maptype = varargin{idx+1};
                case 'alpha'
                    alphaData = varargin{idx+1};
                case 'refresh'
                    autoRferesh = varargin{idx+1};
                case 'showlabels'
                    ShowLabels = varargin{idx+1};
                case 'marker'
                    markerlist{markeridx} = varargin{idx+1};
                    markeridx = markeridx + 1;
                case 'autoaxis'
                    autoAxis = varargin{idx+1};
                case 'apikey'
                    apiKey = varargin{idx+1}; % set new key
                    % save key to file
                    funcFile = which('plot_google_map.m');
                    pth = fileparts(funcFile);
                    keyFile = fullfile(pth,'api_key.mat');
                    save(keyFile,'apiKey')
                otherwise
                    error(['Unrecognized        variable:        '
varargin{idx}])
```

```
            end
        end
    end
    if height > 640
        height = 640;
    end
    if width > 640
        width = 640;
    end

    curAxis = axis;
    % Enforce Latitude constraints of EPSG:900913
    if curAxis(3) < -85
        curAxis(3) = -85;
    end
    if curAxis(4) > 85
        curAxis(4) = 85;
    end
    % Enforce longitude constrains
    if curAxis(1) < -180
        curAxis(1) = -180;
    end
    if curAxis(1) > 180
        curAxis(1) = 0;
    end
    if curAxis(2) > 180
        curAxis(2) = 180;
    end
    if curAxis(2) < -180
        curAxis(2) = 0;
    end

    if isequal(curAxis,[0 1 0 1]) % probably an empty figure
        % display world map
        curAxis = [-200 200 -85 85];
        axis(curAxis)
    end


    if autoAxis
        % adjust current axis limit to avoid strectched maps
        [xExtent,yExtent]      =      latLonToMeters(curAxis(3:4),
curAxis(1:2) );
        xExtent = diff(xExtent); % just the size of the span
        yExtent = diff(yExtent);
        % get axes aspect ratio
        drawnow
```

```
        org_units = get(axHandle,'Units');
        set(axHandle,'Units','Pixels')
        ax_position = get(axHandle,'position');
        set(axHandle,'Units',org_units)
        aspect_ratio = ax_position(4) / ax_position(3);

        if xExtent*aspect_ratio > yExtent
            centerX = mean(curAxis(1:2));
            centerY = mean(curAxis(3:4));
            spanX = (curAxis(2)-curAxis(1))/2;
            spanY = (curAxis(4)-curAxis(3))/2;

            % enlarge the Y extent
            spanY = spanY*xExtent*aspect_ratio/yExtent; % new span
            if spanY > 85
                spanX = spanX * 85 / spanY;
                spanY = spanY * 85 / spanY;
            end
            curAxis(1) = centerX-spanX;
            curAxis(2) = centerX+spanX;
            curAxis(3) = centerY-spanY;
            curAxis(4) = centerY+spanY;
        elseif yExtent > xExtent*aspect_ratio

            centerX = mean(curAxis(1:2));
            centerY = mean(curAxis(3:4));
            spanX = (curAxis(2)-curAxis(1))/2;
            spanY = (curAxis(4)-curAxis(3))/2;
            % enlarge the X extent
            spanX  =  spanX*yExtent/(xExtent*aspect_ratio);  %  new
span
            if spanX > 180
                spanY = spanY * 180 / spanX;
                spanX = spanX * 180 / spanX;
            end

            curAxis(1) = centerX-spanX;
            curAxis(2) = centerX+spanX;
            curAxis(3) = centerY-spanY;
            curAxis(4) = centerY+spanY;
        end
        % Enforce Latitude constraints of EPSG:900913
        if curAxis(3) < -85
            curAxis(3:4) = curAxis(3:4) + (-85 - curAxis(3));
        end
        if curAxis(4) > 85
            curAxis(3:4) = curAxis(3:4) + (85 - curAxis(4));
```

```
        end
        axis(curAxis) % update axis as quickly as possible, before
downloading new image
        drawnow
    end

    % Delete previous map from plot (if exists)
    if nargout <= 1 % only if in plotting mode
        curChildren = get(axHandle,'children');
        map_objs = findobj(curChildren,'tag','gmap');
        bd_callback = [];
        for idx = 1:length(map_objs)
            if ~isempty(get(map_objs(idx),'ButtonDownFcn'))
                % copy callback properties from current map
                bd_callback = get(map_objs(idx),'ButtonDownFcn');
            end
        end
        delete(map_objs)

    end

    % Calculate zoom level for current axis limits
    [xExtent,yExtent] = latLonToMeters(curAxis(3:4), curAxis(1:2)
);
    minResX = diff(xExtent) / width;
    minResY = diff(yExtent) / height;
    minRes = max([minResX minResY]);
    tileSize = 256;
    initialResolution = 2 * pi * 6378137 / tileSize; %
156543.03392804062 for tileSize 256 pixels
    zoomlevel = floor(log2(initialResolution/minRes));

    % Enforce valid zoom levels
    if zoomlevel < 0
        zoomlevel = 0;
    end
    if zoomlevel > 19
        zoomlevel = 19;
    end

    % Calculate center coordinate in WGS1984
    lat = (curAxis(3)+curAxis(4))/2;
    lon = (curAxis(1)+curAxis(2))/2;

    % CONSTRUCT QUERY URL
    preamble = 'http://maps.googleapis.com/maps/api/staticmap';
    location = ['?center=' num2str(lat,10) ',' num2str(lon,10)];
```

```
   zoomStr = ['&zoom=' num2str(zoomlevel)];
   sizeStr = ['&scale=' num2str(scale) '&size=' num2str(width) 'x'
num2str(height)];
   maptypeStr = ['&maptype=' maptype ];
   if ~isempty(apiKey)
       keyStr = ['&key=' apiKey];
   else
       keyStr = '';
   end
   markers = '&markers=';
   for idx = 1:length(markerlist)
       if idx < length(markerlist)
           markers = [markers markerlist{idx} '%7C'];
       else
           markers = [markers markerlist{idx}];
       end
   end
   if ShowLabels == 0
       labelsStr                                                    =
'&style=feature:all|element:labels|visibility:off';
   else
       labelsStr = '';
   end
   if ismember(maptype,{'satellite','hybrid'})
       filename = 'tmp.jpg';
       format = '&format=jpg';
       convertNeeded = 0;
   else
       filename = 'tmp.png';
       format = '&format=png';
       convertNeeded = 1;
   end
   sensor = '&sensor=false';
   url = [preamble location zoomStr sizeStr maptypeStr format
markers labelsStr sensor keyStr];

   % Get the image
   try
       urlwrite(url,filename);
   catch % error downloading map
       warning(sprintf(['Unable to download map form Google
Servers.\n' ...
           'Possible reasons: no network connection, or quota
exceeded.\n' ...
           'Consider using an API key if quota problems
persist.']));
       varargout{1} = [];
```

```
        varargout{2} = [];
        varargout{3} = [];
        return
    end
    [M Mcolor] = imread(filename);
    M = cast(M,'double');
    delete(filename); % delete temp file
    width = size(M,2);
    height = size(M,1);

    % Calculate a meshgrid of pixel coordinates in EPSG:900913
    centerPixelY = round(height/2);
    centerPixelX = round(width/2);
    [centerX,centerY] = latLonToMeters(lat, lon ); % center
coordinates in EPSG:900913
    curResolution = initialResolution / 2^zoomlevel/scale; %
meters/pixel (EPSG:900913)
    xVec = centerX + ((1:width)-centerPixelX) * curResolution; % x
vector
    yVec = centerY + ((height:-1:1)-centerPixelY) * curResolution;
% y vector
    [xMesh,yMesh] = meshgrid(xVec,yVec); % construct meshgrid

    % convert meshgrid to WGS1984
    [lonMesh,latMesh] = metersToLatLon(xMesh,yMesh);

    % We now want to convert the image from a colormap image with
an uneven
    % mesh grid, into an RGB truecolor image with a uniform grid.
    % This would enable displaying it with IMAGE, instead of PCOLOR.
    % Advantages are:
    % 1) faster rendering
    % 2) makes it possible to display together with other colormap
annotations (PCOLOR, SCATTER etc.)

    % Convert image from colormap type to RGB truecolor (if PNG is
used)
    if convertNeeded
        imag = zeros(height,width,3);
        for idx = 1:3
            imag(:,:,idx)       =       reshape(Mcolor(M(:)+1+(idx-
1)*size(Mcolor,1)),height,width);
        end
    else
        imag = M/255;
    end
```

```
   % Next, project the data into a uniform WGS1984 grid
   sizeFactor = 1; % factoring of new image
   uniHeight = round(height*sizeFactor);
   uniWidth = round(width*sizeFactor);
   latVect = linspace(latMesh(1,1),latMesh(end,1),uniHeight);
   lonVect = linspace(lonMesh(1,1),lonMesh(1,end),uniWidth);
   [uniLonMesh,uniLatMesh] = meshgrid(lonVect,latVect);
   uniImag = zeros(uniHeight,uniWidth,3);

   % old version (projection using INTERP2)
   % for idx = 1:3
   %      % 'nearest' method is the fastest. difference from other
methods is neglible
   %                                                uniImag(:,:,idx)    =
interp2(lonMesh,latMesh,imag(:,:,idx),uniLonMesh,uniLatMesh,'nea
rest');
   % end
   uniImag                                                              =
myTurboInterp2(lonMesh,latMesh,imag,uniLonMesh,uniLatMesh);

   if nargout <= 1 % plot map
       % display image
       h = image(lonVect,latVect,uniImag);
       set(gca,'YDir','Normal')
       set(h,'tag','gmap')
       set(h,'AlphaData',alphaData)

       % add a dummy image to allow pan/zoom out to x2 of the image
extent
       h_tmp       =       image(lonVect([1       end]),latVect([1
end]),zeros(2),'Visible','off');
       set(h_tmp,'tag','gmap')

       % older version (display without conversion to uniform grid)
       % h =pcolor(lonMesh,latMesh,(M));
       % colormap(Mcolor)
       % caxis([0 255])
       % warning off % to avoid strange rendering warnings
       % shading flat

       uistack(h,'bottom') % move map to bottom (so it doesn't hide
previously drawn annotations)
       axis(curAxis) % restore original zoom
       if nargout == 1
           varargout{1} = h;
       end
```

```
      % if auto-refresh mode - override zoom callback to allow
autumatic
      % refresh of map upon zoom actions.
      zoomHandle = zoom;
      panHandle = pan;
      if autoRferesh

set(zoomHandle,'ActionPostCallback',@update_google_map);
          set(panHandle,                    'ActionPostCallback',
@update_google_map);
      else % disable zoom override
          set(zoomHandle,'ActionPostCallback',[]);
          set(panHandle, 'ActionPostCallback',[]);
      end

      % set callback for figure resize function, to update extents
if figure
      % is streched.
      figHandle = get(axHandle,'Parent');
      set(figHandle, 'ResizeFcn', @update_google_map_fig);

      % set callback properties
      set(h,'ButtonDownFcn',bd_callback);
  else % don't plot, only return map
      varargout{1} = lonVect;
      varargout{2} = latVect;
      varargout{3} = uniImag;
  end


  % Coordinate transformation functions

  function [lon,lat] = metersToLatLon(x,y)
  % Converts XY point from Spherical Mercator EPSG:900913 to
lat/lon in WGS84 Datum
  originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
  lon = (x ./ originShift) * 180;
  lat = (y ./ originShift) * 180;
  lat = 180 / pi * (2 * atan( exp( lat * pi / 180)) - pi / 2);

  function [x,y] = latLonToMeters(lat, lon )
  % Converts given lat/lon in WGS84 Datum to XY in Spherical
Mercator EPSG:900913"
  originShift = 2 * pi * 6378137 / 2.0; % 20037508.342789244
  x = lon * originShift / 180;
  y = log(tan((90 + lat) * pi / 360 )) / (pi / 180);
  y = y * originShift / 180;
```

```
   function ZI = myTurboInterp2(X,Y,Z,XI,YI)
   % An extremely fast nearest neighbour 2D interpolation, assuming
both input
   % and output grids consist only of squares, meaning:
   % - uniform X for each column
   % - uniform Y for each row
   XI = XI(1,:);
   X = X(1,:);
   YI = YI(:,1);
   Y = Y(:,1);

   xiPos = nan*ones(size(XI));
   xLen = length(X);
   yiPos = nan*ones(size(YI));
   yLen = length(Y);
   % find x conversion
   xPos = 1;
   for idx = 1:length(xiPos)
       if XI(idx) >= X(1) && XI(idx) <= X(end)
           while xPos < xLen && X(xPos+1)<XI(idx)
               xPos = xPos + 1;
           end
           diffs = abs(X(xPos:xPos+1)-XI(idx));
           if diffs(1) < diffs(2)
               xiPos(idx) = xPos;
           else
               xiPos(idx) = xPos + 1;
           end
       end
   end
   % find y conversion
   yPos = 1;
   for idx = 1:length(yiPos)
       if YI(idx) <= Y(1) && YI(idx) >= Y(end)
           while yPos < yLen && Y(yPos+1)>YI(idx)
               yPos = yPos + 1;
           end
           diffs = abs(Y(yPos:yPos+1)-YI(idx));
           if diffs(1) < diffs(2)
               yiPos(idx) = yPos;
           else
               yiPos(idx) = yPos + 1;
           end
       end
   end
```

```
   ZI = Z(yiPos,xiPos,:);


   function update_google_map(obj,evd)
   % callback function for auto-refresh
   drawnow;
   global inputParams
   if isfield(inputParams,['ax' num2str(gca*1e6,'%.0f')])
       params = inputParams.(['ax' num2str(gca*1e6,'%.0f')]);
       plot_google_map(params{:});
   end


   function update_google_map_fig(obj,evd)
   % callback function for auto-refresh
   drawnow;
   global inputParams
   axes_objs = findobj(get(gcf,'children'),'type','axes');
   for idx = 1:length(axes_objs)
       if
~isempty(findobj(get(axes_objs(idx),'children'),'tag','gmap'));
           if                          isfield(inputParams,['ax'
num2str(axes_objs(idx)*1e6,'%.0f')])
               params             =             inputParams.(['ax'
num2str(axes_objs(idx)*1e6,'%.0f')]);
           else
               params = {};
           end
           axes(axes_objs(idx));
           plot_google_map(params{:});
           break;
       end
   end
```

GPS_initilize

```
   function [Lon0,Lat0] = GPS_Initialize
   global Navi;
   fopen(Navi);
   N = 10;
   Latitude0  = 0;
   Longitude0 = 0;
   for i = 1:N
       while (1)
           str         = fscanf(Navi);
           if length(str)>=60
```

```
            if strcmp(str(1:6),'$GPRMC') & strcmp(str(15),'A')
                break;
            end
        end
        pause(0.001);
    end

    year        = str(54:55);
    mo          = str(52:53);
    day         = str(50:51);
    hr          = str(8:9);
    min         = str(10:11);
    sec         = str(12:13);
    hour        = str2num(hr);
    minute      = str2num(min);
    second      = str2num(sec);
    TimeNew    = 3600*hour + 60*minute + second;

    LatiMin    = str(17:18);
    LatiSec    = str(19:20);
    LatiDec    = str(22:25);
    Latitude   = str(27);
    Lat_dd     = str2num(LatiMin);
    Lat_mm     = str2num(LatiSec);
    Lat_dec    = str2num(LatiDec);
    Lat        = Lat_dd + (Lat_mm + Lat_dec/10000)/60;

    LongMin    = str(29:31);
    LongSec    = str(32:33);
    LongDec    = str(35:38);
    Longitude  = str(40);
    Lon_dd     = str2num(LongMin);
    Lon_mm     = str2num(LongSec);
    Lon_dec    = str2num(LongDec);
    Lon        = -(Lon_dd + (Lon_mm + Lon_dec/10000)/60);

    Longitude0  = Longitude0+Lon;
    Latitude0   = Latitude0+Lat
    pause(0.001);
end
Lon0 = Longitude0/N;
Lat0 = Latitude0/N;
fclose(Navi);
end
```

LIDAR_initialize

```
   function [X1,Y1]=LIDAR_Initialize
   global Serial_Port;
   Serial_Port                                                    =
serial('COM4','BaudRate',9600,'InputBufferSize',1024);
   Fid_Display = 1;
   fopen(Serial_Port);
   DataToSend_Continuous_Stop=uint8([2 0 2 0 32 37 53 8]);
   fwrite(Serial_Port,DataToSend_Continuous_Stop);
   DataSet_Stop = fread(Serial_Port);
   Serial_Port.RequestToSend='on';
   % set up baudrate
    DataToSend_Baudrate=[2 0 2 0 32 66 82 8];
    fwrite(Serial_Port,DataToSend_Baudrate);
    BaudRate_Response = fread(Serial_Port);
       if isempty(BaudRate_Response)==1
          Message=['BaudRate: Failed setted'];
          disp(Message);
       else
          Fid_BaudRate=mean((BaudRate_Response==[6 2 128 3 0 160 0
16 22 10]'));
          if Fid_BaudRate==1
             Message=['BaudRate: Succeed setted - 9600 bps'];
             disp(Message);
          else
             Message=['BaudRate: Failed setted'];
             disp(Message);
          end
       end
       Serial_Port.BaudRate=9600;
   % set up distance model
   DataToSend_Installation=uint8([2 0 10 0 32 0 83 73 67 75 95 76
77 83 190 197]);
            fwrite(Serial_Port,DataToSend_Installation);
            Installation_Response=fread(Serial_Port);

            if isempty(Installation_Response)==1
               Message={Message;'Distance Modle: Failed setted'};
               disp(Message);
            else

               Fid_Installation=mean((BaudRate_Response==[6 2 128 3
0 160 0 16 22 10]'));
               if Fid_Installation==1
                   %%%%%%%%%%mm model Set Up
```

```
                DataToSend_Distance_Model= uint8([2 0 33 0 119 0
0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 232
114]);
                fwrite(Serial_Port,DataToSend_Distance_Model);
                Distance_Model_Response=fread(Serial_Port);
                if isempty(Installation_Response)==1
                    Message={Message;'Distance   Modle:   Failed
setted'};
                    disp(Message);
                else

Fid_Distance_Modle=mean((Distance_Model_Response==[6  2  128  37  0
247 0 0 0 70 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 2 114 16 156 103]'));
                    if Fid_Distance_Modle==1;
                        Message={Message;'Distance       Modle:
Succeed setted - cm Model'};
                        disp(Message);
                    else
                        Message={Message;'Distance       Modle:
Failed setted'};
                        disp(Message);
                    end
                end
            else
                Message={Message;'Distance    Modle:    Failed
setted'};
                disp(Message);
            end
        end
  % set up angleresolution
   DataToSend_AngleRange_Resolution=uint8([2 0 5 0 59 180 0 50 0
59 31]);
        fwrite(Serial_Port,DataToSend_AngleRange_Resolution);
        AngleRange_Resolution_Response=fread(Serial_Port);
        if isempty(AngleRange_Resolution_Response)==1
            Message={Message{1};Message{2};'Range  Resolution:
Failed setted'};
            disp(Message);
        else

Fid_AngleRange_Resolution=mean((AngleRange_Resolution_Response==
[6 2 128 7 0 187 1 180 0 50 0 16 3 157]'));
            if Fid_AngleRange_Resolution==1
                Message={Message{1};Message{2};'Range
Resolution: Succeed setted - 180/0.5'};
                disp(Message);
```

```
            else
                Message={Message{1};Message{2};'Range
Resolution: Failed setted'};
                disp(Message);
            end
        end
  % read data and display it
  DataSet=[];
  DataToSend_Continuous=uint8([2 0 2 0 32 36 52 8]);

  fwrite(Serial_Port,DataToSend_Continuous);
  figure,axes;
  X1       = [];
  Y1       = [];
  Data_Continuous_Head=fread(Serial_Port,10);
  % initialize-- find stationary objects
     for i = 1:30
          Data_Continuous = fread(Serial_Port,732);
                pause(0.01);
                DataSet=[DataSet,Data_Continuous'];

Data_C=Data_Continuous(8:length(Data_Continuous)-3,:);

                Data_C_Bin=dec2bin(Data_C);
                Length_Vaild=length(Data_C_Bin);
                for i=1:2:Length_Vaild
                    Temp=Data_C_Bin(i,:);
                    Data_C_Bin(i,:)=Data_C_Bin(i+1,:);
                    Data_C_Bin(i+1,:)=Temp;
                end

                for i=1:Length_Vaild/2
                    Data_Continuous_Bin(i,:)=[Data_C_Bin(2*i-
1,:) Data_C_Bin(2*i,:)];
                end

Data_Continuous_Dec=bin2dec(Data_Continuous_Bin);

                Angle_Space=linspace(0*pi/180,180*pi/180,361);
                ROI_Range=8000;

ROI_Index=find(Data_Continuous_Dec<=ROI_Range);

[X,Y]=pol2cart(Angle_Space(ROI_Index),Data_Continuous_Dec(ROI_In
dex)');
                X1 = [X1 X];
                Y1 = [Y1 Y];
```

```
    end
end
```

Convert_GPS_to_LIDAR

```
function [X0,Y0] = ConvertGPS2LIDAR(X1,Y1)
global Serial_Port;
for kkk = 1:10
DataSet = [];
Data_Continuous = fread(Serial_Port,732);
pause(0.01);
DataSet=[DataSet,Data_Continuous'];
Data_C=Data_Continuous(8:length(Data_Continuous)-3,:);

Data_C_Bin=dec2bin(Data_C);
Length_Vaild=length(Data_C_Bin);
for i=1:2:Length_Vaild
    Temp=Data_C_Bin(i,:);
    Data_C_Bin(i,:)=Data_C_Bin(i+1,:);
    Data_C_Bin(i+1,:)=Temp;
end

for i=1:Length_Vaild/2
    Data_Continuous_Bin(i,:)=[Data_C_Bin(2*i-1,:)
Data_C_Bin(2*i,:)];
end
Data_Continuous_Dec=bin2dec(Data_Continuous_Bin);

Angle_Space=linspace(0*pi/180,180*pi/180,361);
ROI_Range=8000;
ROI_Index=find(Data_Continuous_Dec<=ROI_Range);
[X,Y]=pol2cart(Angle_Space(ROI_Index),Data_Continuous_Dec(ROI_
Index)');

N   = length(X);
Show_Index=[];

for i = 1:N
    Distance=sqrt((X1-X(i)).^2+(Y1-Y(i)).^2);
    if isempty(find(Distance<=15))
        Show_Index = [Show_Index i];
    end
end
X_s = X(Show_Index);
Y_s = Y(Show_Index);
```

```
if ~isempty(X_s)
    X0 = mean(X_s);
    Y0 = mean(Y_s);
    %Dist = sqrt((XNew-XWay)^2+(YNew-YWay)^2);
else
    disp('Error:Robot not found');
end
end
end
```

GPS_to_LIDAR

```
function [X,Y] = GPS2LIDAR(Lon,Lat,Lon0,Lat0,X0,Y0)
a  = 6378137.0;
b  = 6356752.3142;
pi = 3.1416;
if (Lon==-1)&(Lat==-1)
    X = -1;
    Y = -1;
else
    % Distance corresponding to 1 deg change in Latitude
    F_lon    =    (pi/180)    *    a^2*cos(Lat/180*pi)        /
(a^2*(cos(Lat/180*pi))^2 + b^2*(sin(Lat/180*pi))^2)^(1/2);
    % Distance corresponding to 1 deg change in Longitude
    F_lat = (pi/180) * a^2*b^2   /   (a^2*(cos(Lat/180*pi))^2 +
b^2*(sin(Lat/180*pi))^2)^(3/2);
    Dlat  = F_lon*(Lat-Lat0)*100; % (unit--cm)
    Dlon  = F_lat*(Lon-Lon0)*100; % (unit--cm)
    X     = X0+Dlon;
    Y     = Y0+Dlat;
end
end
```

Read_GPSXbee_new4

```
function [Lon,Lat,Heading] = readGPS_xbee_new4(Lspeed,Rspeed)
global Navi;

while (1)
    str         = fscanf(Navi);
    if length(str)>=60
        if strcmp(str(1:6),'$GPRMC') & strcmp(str(15),'A')
            break;
        end
    else if (length(str)<=13) & (length(str)>=8)
```

```
            if strcmp(str(1:4),'None')
                break;
            end
        end
    end
end
% send out Lspeed&Rspeed
if (Lspeed >=0) & (Rspeed>=0)
    fprintf(Navi,'%s', 'a');
    disp('a');
elseif Lspeed>0 & Rspeed<0
    fprintf(Navi,'%s','b');
    Rspeed = -Rspeed;
    disp('b');
elseif Lspeed<0 & Rspeed>0
    fprintf(Navi,'%s','c');
    Lspeed=-Lspeed;
    disp('c');
end

fprintf(Navi,'%d\n',Lspeed);
fprintf(Navi,'%d\n',Rspeed);

disp('Lspeed = ')
disp(Lspeed)

disp('Rspeed = ')
disp(Rspeed)

% Read NMEA string
if (length(str)>=60)
    year        = str(54:55);
    mo          = str(52:53);
    day         = str(50:51);
    hr          = str(8:9);
    min         = str(10:11);
    sec         = str(12:13);
    hour        = str2num(hr);
    minute      = str2num(min);
    second      = str2num(sec);
    TimeNew    = 3600*hour + 60*minute + second;

    LatiMin    = str(17:18);
    LatiSec    = str(19:20);
    LatiDec    = str(22:25);
    Latitude   = str(27);
    Lat_dd     = str2num(LatiMin);
```

```
      Lat_mm     = str2num(LatiSec);
      Lat_dec    = str2num(LatiDec);
      Lat        = Lat_dd + (Lat_mm + Lat_dec/10000)/60;

      LongMin    = str(29:31);
      LongSec    = str(32:33);
      LongDec    = str(35:38);
      Longitude  = str(40);
      Lon_dd     = str2num(LongMin);
      Lon_mm     = str2num(LongSec);
      Lon_dec    = str2num(LongDec);
      Lon        = -(Lon_dd + (Lon_mm + Lon_dec/10000)/60);

      Head       = str(61:end);
      Heading    = str2num(Head);

      disp([hr ':' min ':' sec '     UTC']);
      disp([day '/' mo '/' year '     DDMMYY']);
      disp([Latitude '    ' LatiMin ' DEG ' LatiSec '.' LatiDec '
MIN']);
      disp([Longitude '   ' LongMin ' DEG ' LongSec '.' LongDec '
MIN']);
      disp([str]);
      fprintf('\r');
  else if length(str)<=13
          Lat     = -1;
          Lon     = -1;
          Head    = str(5:end);
          Heading = str2num(Head);
          TimeNew = -1;
      end
  end

  end
```

LIDAR_communication

```
   function
[Dist,HeadWay,X_collect,Y_collect,XNew,YNew]=LIDAR_Communication
(X1,Y1,X_gps,Y_gps,XWay,YWay,X_collect0,Y_collect0,XOld,YOld,Hea
dWayOld,DistOld)
  global Serial_Port;
  DataSet = [];
  Data_Continuous = fread(Serial_Port,732);
  pause(0.01);
  DataSet=[DataSet,Data_Continuous'];
```

```
   Data_C=Data_Continuous(8:length(Data_Continuous)-3,:);

   Data_C_Bin=dec2bin(Data_C);
   Length_Vaild=length(Data_C_Bin);
   for i=1:2:Length_Vaild
       Temp=Data_C_Bin(i,:);
       Data_C_Bin(i,:)=Data_C_Bin(i+1,:);
       Data_C_Bin(i+1,:)=Temp;
   end

   for i=1:Length_Vaild/2
       Data_Continuous_Bin(i,:)=[Data_C_Bin(2*i-1,:)
Data_C_Bin(2*i,:)];
   end
   Data_Continuous_Dec=bin2dec(Data_Continuous_Bin);

   Angle_Space=linspace(0*pi/180,180*pi/180,361);
   ROI_Range=8000;
   ROI_Index=find(Data_Continuous_Dec<=ROI_Range);
   [X,Y]=pol2cart(Angle_Space(ROI_Index),Data_Continuous_Dec(ROI_
Index)');

   N   = length(X);
   Show_Index=[];

   for i = 1:N
       Distance=sqrt((X1-X(i)).^2+(Y1-Y(i)).^2);
       if isempty(find(Distance<=15))
           Show_Index = [Show_Index i];
       end
   end
   X_s = X(Show_Index);
   Y_s = Y(Show_Index);

   if ~isempty(X_s)
       plot(X_s,Y_s,'*');
       axis([-2000,2000,0,5000]);
       grid on;
       if (X_gps~=-1)|(Y_gps~=-1)
           DistXY  = sqrt((X_s - X_gps).^2+(Y_s - Y_gps).^2);
           IndexXY = find(DistXY<1500);
           X_s     = X_s(IndexXY);
           Y_s     = Y_s(IndexXY);
       end
       XNew = mean(X_s);
       YNew = mean(Y_s);
       Dist = sqrt((XNew-XWay)^2+(YNew-YWay)^2);
```

```
      % Calculate HeadWay-----------------------------------
      if XWay == XNew
          if YWay < YNew
              HeadWay = 180;
          elseif YWay > YNew
              HeadWay = 0;
          else
              HeadWay = 0;
          end
      end

      if YNew == YWay
          if XNew < XWay          %Moving West
              HeadWay = 90;
          elseif XNew > XWay
              HeadWay = 270;            % Moving East
          else
              HeadWay = 0;            % Stationary
          end
      end
      % 1st quadrant
      if ((XWay> XNew) & (YWay > YNew))
          HeadWay  =  180/pi*tanh(abs(XNew  -  XWay)  /  abs(YWay  -
YNew));
      end

      % 2nd quadrant
      if ((XWay > XNew) & (YWay < YNew))
          HeadWay = 90 + 180/pi*tanh(abs(YNew - YWay) / abs(XNew
- XWay));
      end

      % 3rd quadrant
      if ((XWay < XNew) & (YWay < YNew))
          HeadWay = 180 +  180/pi*tanh(abs(XWay - XNew) / abs(YNew
- YWay));
      end

      % 4th quadrant
      if ((XWay < XNew) & (YWay > YNew))
          HeadWay = 270 + 180/pi*tanh(abs(YWay - YNew) / abs(XWay
- XNew));
      end
      %-----------------------------------------------------------
-------------------------------------------
```

```
   %Heading = 0;
       %-------------------------------------------------------------
---------------------------------
       X_collect = [X_collect0 X_s];
       Y_collect = [Y_collect0 Y_s];
   else
       disp('NO Moving Object');
       Dist       = DistOld;
       HeadWay    = HeadWayOld;
       X_collect  = X_collect0;
       Y_collect  = Y_collect0;
       XNew       = XOld;
       YNew       = YOld;
   end

   end
```

LIDAR_read

```
   function [X_lidar,Y_lidar]=LIDAR_read(X1,Y1)
   global Serial_Port;

       DataSet = [];
       Data_Continuous = fread(Serial_Port,732);
       pause(0.01);
       DataSet=[DataSet,Data_Continuous'];
       Data_C=Data_Continuous(8:length(Data_Continuous)-3,:);

       Data_C_Bin=dec2bin(Data_C);
       Length_Vaild=length(Data_C_Bin);
       for i=1:2:Length_Vaild
           Temp=Data_C_Bin(i,:);
           Data_C_Bin(i,:)=Data_C_Bin(i+1,:);
           Data_C_Bin(i+1,:)=Temp;
       end

       for i=1:Length_Vaild/2
           Data_Continuous_Bin(i,:)=[Data_C_Bin(2*i-1,:)
Data_C_Bin(2*i,:)];
       end
       Data_Continuous_Dec=bin2dec(Data_Continuous_Bin);

       Angle_Space=linspace(0*pi/180,180*pi/180,361);
       ROI_Range=8000;
       ROI_Index=find(Data_Continuous_Dec<=ROI_Range);
```

```
[X,Y]=pol2cart(Angle_Space(ROI_Index),Data_Continuous_Dec(ROI_In
dex)');

      N   = length(X);
      Show_Index=[];

      for i = 1:N
          Distance=sqrt((X1-X(i)).^2+(Y1-Y(i)).^2);
          if isempty(find(Distance<=15))
              Show_Index = [Show_Index i];
          end
      end
      X_s = X(Show_Index);
      Y_s = Y(Show_Index);

      if ~isempty(X_s)
          plot(X_s,Y_s,'*');
          axis([-2000,2000,0,5000]);
          grid on;
          X_lidar = mean(X_s);
          Y_lidar = mean(Y_s);
      else
          disp('NO Moving Object');
          X_lidar = -1;
          Y_lidar = -1;
      end

  end
```