Final Report

Mobile Automated Directory
M.A.D. “Max”

Submitted By:
Group 1
Cory Royal
Curtis Dylan Odle
William Scott Morris

Submitted To:
Instructor - August Allo
EE 4813 - Design II
The University of Texas at San Antonio
1 UTSA Circle, San Antonio, TX 78249

April 24, 2015

1.0 Table of Contents

1.0 Executive Summary

The uses of map directories in malls and other large area locations have proved beneficial in their respective venues for locating a desired destination. These directories give a visual view of where your current location is on a map and a listing of all the possible destinations in which you may want to go. The usefulness of these simple accessories is great. Group 1, which consists of electrical engineering students Cory Royal, Curtis Dylan Odle, and William Scott Morris, has designed a more interactive and possibly effective means for people to find their desired destination for a more specific application; locating key interest points on the UTSA main campus. We have accomplished this task in our project by using a mobile robot retrofitted with various locating and heading equipment. All aspects to the design of this automated mobile directory were investigated and pursued in order to result in a final working prototype. The design project that our group has assembled is a proof of concept and is intended to be improved upon.

As proposed, the device was intended to consist of the robot which included a Microsoft Kinect sensor, Kobuki mobile base, and Netbook. The initial device also called for a touchscreen and display to be used for possible user inputs as well as map and route presentation and give an interactive face, as well as speakers for the robot to talk back to the user. GPS was also the decided on approach for traversing the UTSA campus. From our initial plan, we have trimmed components as well as added some as necessary to fulfill the need of a working prototype. The components that we did not use in our prototype include the touchscreen and display as well as the speakers. These were excluded due to the need of time and resources necessary elsewhere in the project. We did however need to add a compass module to give heading accuracy we could not retrieve originally.

2.0 Introduction

The UTSA campus can be difficult to get around if you are unfamiliar with the building names, or the layout of the campus. UTSA Engineering Professor Dr. Pranav Bhounsule, wanted to promote UTSA's great and upcoming Engineering Program, and also solve a problem at the same time. A Mobile Automated Directory that interacts with people is a great way to achieve these two goals. M.A.D. "Max" attempts to fulfil these goals as well as promote UTSA Engineering awareness to both students and non-students.

"Max" is equipped with a GPS module, a compass, and a Microsoft Kinect Sensor (visual sensor), microphone, and voice to text software, to help it get its job done. The user will simply speak to "Max" and state a destination where they would like to go, and if the destination is somewhere that "Max" has in a list of buildings known to the UTSA campus, "Max" will then move to that location. The user then just has to follow "Max" to get to their designated building.

The prototype was designed following specific design constraints, requirements, and specifications designed around a product that would successfully advertise UTSA's engineering program on campus. The design constraints, requirements, and specifications were developed and revised through research, consultations with mechanical engineering professionals, and the general public. The engineering design team's attention to detail in all aspects of the project allowed for timely completion of an operational prototype. The team invoked the use of Microsoft Project, an open-source project management/scheduling computer application, to store tasks, set milestones, and monitor progress throughout the entire project.

Hardware	**101 days**	**Mon 12/1/14**	**Fri 4/17/15**
Order Turtlebot, Netbook, GPS module, Arduino	2 days	Mon 12/1/14	Tue 12/2/14
Await Ordered Parts	27 days	Wed 12/3/14	Thu 1/8/15
Combine Turtlebot Hardware	5 days	Fri 1/9/15	Thu 1/15/15
GPS Install	**8 days**	**Fri 1/9/15**	**Tue 1/20/15**
Connect GPS module to Arduino	5 days	Fri 1/9/15	Thu 1/15/15
Attach GPS antenna	3 days	Fri 1/16/15	Tue 1/20/15
Arduino Power Modification	**4 days**	**Tue 4/14/15**	**Fri 4/17/15**
Acquire voltage regulator and appropriate power connectors	1 day	Tue 4/14/15	Tue 4/14/15
Build Arduino power plug incorporating voltage regulator	3 days	Wed 4/15/15	Fri 4/17/15
Hardware Complete	1 day	Wed 1/21/15	Wed 1/21/15
Software	**104 days**	**Mon 12/1/14**	**Wed 4/22/15**
Familiarize ourselves with programming language	31 days	Mon 12/1/14	Mon 1/12/15
Turtlebot movement/Route Creation	**74 days**	**Mon 1/12/15**	**Wed 4/22/15**
Make locations an input	7 days	Mon 1/12/15	Tue 1/20/15
Create movement commands to drive base	10 days	Mon 1/12/15	Fri 1/23/15
Test for correct movement	2 days	Sat 1/24/15	Mon 1/26/15
Use inputs to initiate route creation	5 days	Mon 1/12/15	Fri 1/16/15
Format control so that Turtlebot goes to one waypoint at a time	14 days	Tue 1/27/15	Fri 2/13/15
Integrate waypoints in a complete route to destination	43 days	Mon 2/16/15	Wed 4/15/15
Test for correct choice in route	5 days	Thu 4/16/15	Wed 4/22/15
GPS Integration	**38 days**	**Fri 1/30/15**	**Tue 3/24/15**
Test initial data output	3 days	Fri 1/30/15	Tue 2/3/15
Receive current latitude and longitude from GPS	14 days	Wed 2/4/15	Mon 2/23/15
Integrate GPS data into robot control	21 days	Tue 2/24/15	Tue 3/24/15
Audio Inputs and Feedback	**58 days**	**Fri 1/30/15**	**Tue 4/21/15**
Receive audio inputs	5 days	Fri 1/30/15	Thu 2/5/15
Provide library of audio commands and build supporting code	27 days	Fri 2/6/15	Mon 3/16/15
Test audio commands	6 days	Tue 3/17/15	Tue 3/24/15
Provide Audio feedback to user	8 days	Wed 3/25/15	Fri 4/3/15
Test for correct feedback to user input	7 days	Mon 4/6/15	Tue 4/14/15
Integrate complete speech to text program into robot control	5 days	Wed 4/15/15	Tue 4/21/15
Collision Avoidance	**74 days**	**Mon 1/12/15**	**Wed 4/22/15**
Program abililty to recognize obstacles	17 days	Mon 1/12/15	Mon 2/2/15
Program to stop before obstacle and wait until clear	25 days	Tue 2/3/15	Mon 3/9/15
Integrate into robot control	3 days	Tue 3/10/15	Thu 3/12/15
Test collision avoidance	29 days	Fri 3/13/15	Wed 4/22/15
Software Complete	1 day	Wed 4/22/15	Wed 4/22/15
Project Complete	1 day	Wed 4/22/15	Wed 4/22/15

Fig 1. Project Schedule

In order to complete the project within the given time constraints and organize design phases, the team selected a project manager, Cory Royal. The dynamic and friendship of the team members allowed for a great working environment, and adequate rapport.

3.0 Need Being Addressed

UTSA is a pretty hard campus to transverse if it is your first time here. All the buildings look alike, and have been renamed quite a few times recently, making things all the more difficult to get around campus for someone who doesn't know their way around campus. M.A.D. "Max" makes it easier for new people to transverse the UTSA campus. "Max" will be deployed on the UTSA campus and interact with people attempting to find a certain building on campus.

With "Max" the UTSA campus will be mapped and traversed with ease. A small database of possible locations will be established. This list will include acceptable building names, past building names, as well as the new locations on campus. Having voice commands and having users following him around campus to their needed destinations will make the process enjoyable and easily done. This will be an impressive feat and shed light on the UTSA Engineering program, which is always wanted.

4.0 Literature and Patent Search Results

As soon as we were presented with the idea to design an autonomous robot by Professor Pranav we began coming up with a basic idea for M.A.D "Max". Once the initial idea was established we began our research into the optimal processes/devices that could be utilized. The Patent Search website proved to be difficult as it was hard to find other patents that pertained to our project specifically. One of our biggest problems, when it came to finding patents, was the fact that our project was predominantly writing code and we decided to utilize Open-Source Code. However, a few patents were found which solved some problems that our project was having.

First, we had to familiarize ourselves with the Linux system Ubuntu which M.A.D. "Max" utilizes. Once we had download Ubuntu onto our Workstation we began working with Robot Operating System (ROS). In order to understand how ROS worked we utilized the ROS

wiki Tutorials. This website gave us the required background we needed in order to operate "Max". The website also gave a very small insight into how to write code for ROS. Another PDF that we found on the internet is titled "A Gentle Introduction to ROS" [5] and was used to help us better understand how to format and construct code using ROS.

Our next step involved finding a proper routing system which could effectively navigate through the school. During our research through the Patent Search website we came across the patent called: "System and a method for enabling a vehicle to track a preset path". This Patent described the idea of using a GPS system which is what we decided to pursue. We decided that this was our best option for creating a path system throughout the school.

Finally, we had to find a way to use voice commands from the user in order for "Max" to understand where the user is trying to go. Through our research we found the Open-Source Code called Voce. The code was found on the Voce website. [6] Voce however was too advanced for what we required but it gave us the idea to utilize the same program called Sphinx4. Using Sphinx4 [7] was our best option because as was stated earlier it was Open-Source and was well documented, which allowed us to easily install and begin implementation with the rest of our code.

5.0 Marketing Analysis and Market Strategy

Directories are used in many different applications. Their purpose is to make life easier on the user and help guide them to the point of interest. Examples of the use of these tools can be seen in malls and amusement parks. One can see the want for a more interactive and entertaining means to locate your desired destination.

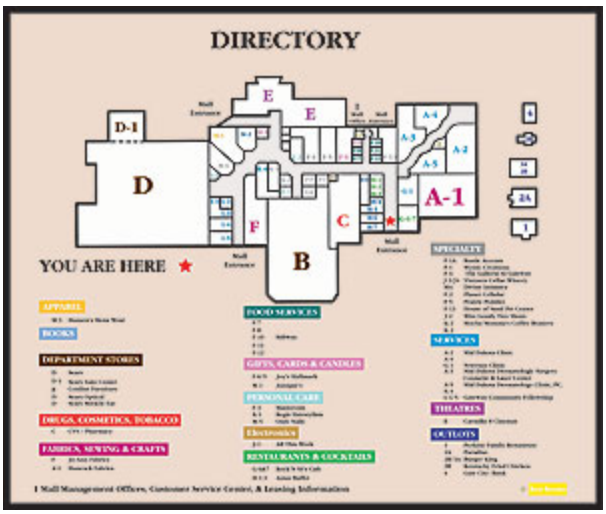

Fig 2. Example of Mall Directory We Aim to Replace

5.1 Large Venues with Multiple Points of Interest

It is large venues that we found will be our target market, along with educational robotics. Because our robot adds flare and ingenuity to the rather simple directory currently in use at such locations, we feel as it can be seen as a replacement or addition to these locations. The purpose of the specific application of our project, “Max”, also entails its additional market; a directory for large university campuses.

5.2 Universities and Education

Robot manufacturing companies would find a complete waypoint traversing system a sought after commodity in educational purposes. Due to the implementation and integration of each separate piece, educational applications are numerous. Many other robots have been manufactured for this same reason. Universities and trade schools would be willing to pay for

such a robot in order to advance the system integration and computer science concepts taught in their institution.

6.0 Engineering Design Constraints

When designing a product to be used, we as engineers have to abide by certain rules. These rules limit our designs for the better good. To incorporate these constraints, while conception of the idea for this project was fresh, our group considered all the global and local design constraints that may affect the design of our project. The constraints considered and how they pertain to our project are listed below.

6.1 Global Design Constraints

Sustainability:

This constraint applies to our design. The utility of our design will be needed/wanted in the future. Because the design is ahead of its time and also one of the first in its category, the sustainability seems to be securely in our favor.

Manufacturability:

This constraint applies to our design. The Turtlebot 2 is already a largely manufactured robot. The other components such as the Arduino Mega 2560, the GPS module, and the magnetometer sensor are all also largely manufactured. Due to the components being manufactured already, the complete package will be easily manufactural, but will need to be ordered. The design does however rely on the continual manufacture of all these parts.

Ethical Considerations:

This constraint does not apply to our design. There is no dilemma with our product and the engineering code of ethics.

Engineering Codes and Standards:

This constraint applies to our design. As all designs do, we also have to comply with engineering codes and standards. We intend to use this product in a university setting or even at malls and amusement parks which means we feel that the product is safe for the public.

Economic Factors:

This constraint applies to our design. Although the purchase of a robotic automated directory can be pricey, given the setting in which this product is intended to be used, we feel such venues have the means of purchasing such a device as they are most often well-funded.

Environmental Effects:

This constraint does not apply to our design. The mobile automated directory is in itself, entirely electrical. No pollution or otherwise harmful conditions are exposed to the environment due to our product.

Health and Safety Issues:

This constraint applies slightly to our design. We have very little safety and health issues with the robot. The only foreseeable safety hazard is that the robot may hit a person while moving. However such precautions such as collision avoidance have been implemented to take away such hazards.

Social Ramifications:

This constraint does not apply to our design. The mobile automated directory is intended to be implemented in a helpful and utilitarian sort of way.

Political Factors:

This constraint does not apply to our design. There are no political ties to our project. It is to be used in a very specific way that in no way causes controversy.

Legal Issues:

This constraint applies to our design. As with all designs, our project can be involved in legal matters for seen or unseen events. As it is an automated robot, our design can be tampered with as well as possibly give unintended results. The possibility of patent infringement and the wrongful disposal of its parts are other examples of possible legal issues this design can result in.

6.2 Local Design Constraints

Cost:

This constraint applies to our design. The cost of the design will large as the robot used alone is expensive. However, as depicted before the venue (universities/malls/amusement parks) in which it is intended to be distributed to have the means of acquiring such a device.

Schedule:

This constraint applies to our design. Schedule is a common constraint in products. Our product will be no exception as it is to be finished before the semester is over for the groups design course it was designed for.

Manufacturability:

This constraint applies to our design. As said above in global constraints, the device relies on the manufacture of other devices such as the Turtlebot 2, GPS module, and magnetometer.

Engineering Codes and Standards:

This constraint applies to our design. All engineering codes and standards must be followed in a local manner just as they should a global manner.

Ethical Considerations:

This constraint does not apply to our design. Again, the product will not trespass on ethical grounds and is intended to be used on a wanted basis. One can choose not to use our product.

Health and Safety Issue:

This constraint applies slightly to our design. Just as in the global design constraints, our product will hardly cause an unsafe environment. Collision avoidance is countering the one small unsafe situation the robot can encounter.

Legal:

This constraint applies to our design. As with all designs, our project can be involved in legal matters for seen or unseen events. As it is an automated robot, our design can be tampered with as well as possibly give unintended results. The possibility of patent infringement and the wrongful disposal of its parts are other examples of possible legal issues this design can result in.

7.0 Product Requirements/Specifications

Requirements involving handling as well as necessary items and features of our mobile automated device are listed and categorized. The list was compiled in the beginning stages of our project development.

7.1 User Requirements

1. The device shall be rechargeable
2. The device shall have a simplistic user interface.
3. The device shall work in a pace at which the average person can keep up with.
4. A collision avoidance system shall be implemented

7.2 System Requirements

1. Must use GPS data.
2. Shall be able to reuse after first use.
3. The product shall be able to be remotely started/stopped.
4. All devices must fit and be securely fashioned onto the Turtlebot 2 chassis.
5. The robot must be able to be powered long enough for a full scale trip to its farthest destination possible.
6. The robot must stay in between the walkways provided.
7. The robot should be able to take minimal abuse and continue to operate.

7.3 Interface Requirements

1. The system shall show all possible destinations
2. The system shall allow for audio commands and/or manual keyboard entry.

8.0 Engineering Codes and Standards

Engineering Codes and Standards were highly considered in the design of our project. The codes and standards referenced are listed below and encompass the guidelines to which our design follow.

IEEE 802.11: Wireless LAN Technology. A set of physical layer standard for implementing wireless local area network computer communication in the 2.4,3.6,5 and 60GHz frequency band. [8]

IEEE 1872-2015 Approved Draft Standard for Ontologies for Robotics and Automation [9]

9.0 Design Concepts

While accounting for other considerations such as design constraints as well as practicality, we had brainstormed several different design concepts. These different design concepts had their differences in the way that we would be locating ourselves on the UTSA campus. We had to do this because; how would you know where you are going if you do not know the destination, or even where you are at? During our thought process, we decided to consider three different possible locating processes. These included GPS (Global Positioning System), Location Pinging, and Kinect Mapping.

9.1 Global Positioning System

Our first concept of GPS involved a GPS sensor that would collect GPS data from several satellites orbiting the Earth. We found after some time in research we could get a GPS module that gives an accuracy of within 3 meters. With the coordinates retrieved we could calculate with simple triangular calculations to move our robot as needed. However, when implemented, the movement was done not through calculations but as hard coded points, due to the inaccuracies in the GPS data.

9.2 Location Pinging

Another concept that we found to be a solid candidate for our project was locations pinging. This concept would involve several sensors placed throughout the UTSA campus as well as another sensor on the robot itself. The sensor on the robot would ping out radially and

when the sensors located on the campus retrieved them, they would give a distance between that sensor and the robot. With this information, once the distances for three or more sensors were found, we could triangulate our position based on the known location of the stationary sensors on campus. Then calculation to our destination would be found.

9.3 Kinect Mapping

Our final concept was established with the preliminary research of the Turtlebot we were to use. Many other people using this robot have performed navigation through an already found procedure of mapping an area with either 2D or 3D with the Microsoft Kinect already on the Turtlebot known as SLAM. This has been done largely indoors to find locations in a room or hallway. With the map created by the Kinect, one could traverse an area without colliding into walls or stationary objects.

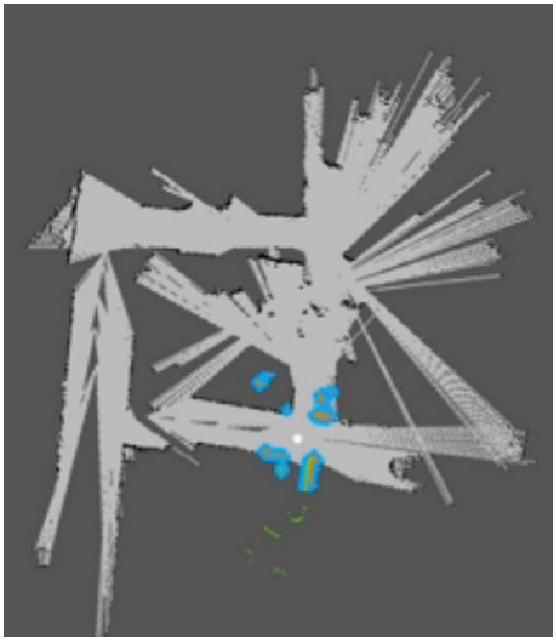

Fig 3. Example of SLAM mapping system [18]

9.4 Pugh Matrix

To determine the concept we felt was best for our project, we used a Pugh matrix. We selected 12 of the most important requirements of our project and assigned a weighting of each requirement as it pertains to our individual needs of the project as a whole. The weighting was from 1 to 10 with 10 meaning the requirement was really important to our design. We then used this weighting for all three of our design concepts to acquire the best suit to our project.

Pugh Matrix				
Design Requirements	Weighting	GPS	Location Pinging	Kinect Mapping
Accuracy	9	8	9	8
Reliability	8	8	5	6
No extra equipment required	7	8	3	10
Minimum preliminary data acquisition	7	5	9	1
Works outside	9	9	9	3
Creation of route	10	10	7	5
Ability to follow route	10	8	7	5
Ease in finding current location	9	10	8	5
Placement of landmarks on map	9	10	9	5
Maintenance	5	9	3	9
Schedule	9	10	6	3
Cost	8	5	5	10
TOTAL	100	843	688	566

Fig 4. Pugh Matrix

From the Pugh matrix formed above, one can see that the GPS concept won in terms of meeting our needs as a project. The accuracy would have been similar in all three cases. Being able to create a route and then being able to follow that route was of great importance to our group. When thinking of these two requirements we found problems in the other two design concepts. For one, the location pinging would involve the complexity of having stored a location on a virtual map of all the destinations as well as constantly reiterating the process of pinging for you location, which if was inaccurate for any reason would take you on a drastically different

path. The Kinect mapping concept was not feasible in the fact that it would involve an extensively large amount of memory and computing time for the Kinect to map such a large area as a university campus. So in turn, the GPS concept won the bid due to its simplistic use and ease of route creation/following.

10.0 High Level Block Diagram

When the project began, we needed an overview of what the project would entail so that we could reference where we were in the project at all times and see the desired result. To do this, our team designed a high level block diagram to demonstrate each portion of the project.

This diagram first consisted of every hardware unit and data output that pertains to each device, however as the semester went on, the block diagram changed as our project changed. For example, the magnetometer was added in and the touchscreen and display were taken out. The final high level block diagram is depicted below.

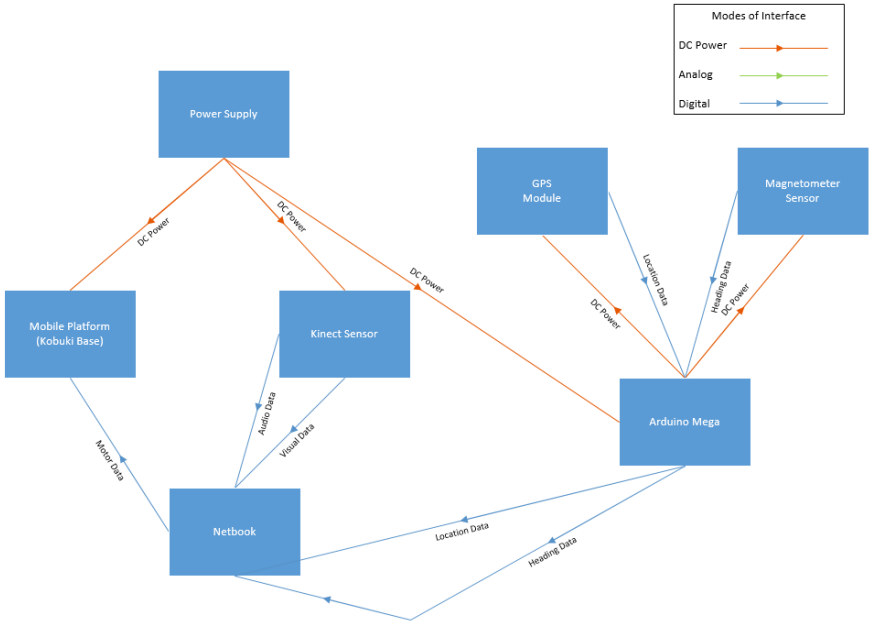

Fig 5. High Level Block Diagram

11.0 Major (Critical) Components

Some major components used in our project were the GPS module, the magnetometer sensor, and the Arduino Mega. The Turtlebot 2 could be considered a major component, however it was decided for us by our sponsor Dr. Pranav Bhounsule.

The GPS module, as described in the design concepts portion of this text, was our best decision in the means of locating our current position and using this location to form a route to our destination. It was simple to use and because we found the Adafruit Ultimate GPS Breakout, we had a device that was advertised to give an acceptable accuracy of three meters.

The magnetometer sensor was a crucial part to our design as it allowed us to give a more precise heading to our mobile robot. It also eliminated the inaccuracies of the motor velocities in the Kobuki base of the Turtlebot.

Finally, the Arduino Mega was a solid choice in our design as it had multiple in/out serial ports in which we could attach both the GPS module as well as the magnetometer sensor. This gave a simplified design integration between the devices in use. The Arduino Mega also gave our project the added benefit of having a decent amount of computing so as to free our netbook of unnecessary computation as it was already intended to run our programs.

12.0 Detailed Design

This section will talk about the “meat and potatoes” of the project and will discuss how the culmination of each separate piece of hardware uses software that integrates and controls it.

12.1 Hardware

There are several aspects regarding the hardware involved in the mobile automated directory. The hardware included are as follows and will be discussed further in the following sections: Turtlebot 2, Adafruit Ultimate GPS Breakout, Magnetometer sensor, Arduino Mega, Wireless Router. With some knowledge of the Arduino microprocessor family, Dylan was tasked with the completion of the GPS and Arduino integration so that we may receive serial output from the device. After knowing we will have needed a magnetometer sensor, Cory began a similar approach to integrate the device alongside the GPS module on the Arduino.

12.1.1 Turtlebot 2

The robot that was used, is the Turtlebot 2, and was purchased from our sponsor Dr. Pranav Bhounsule. It has several different devices that make for a whole package robot. The robot includes a Microsoft Kinect, Kobuki base, and Netbook. As it is a package and comes ready to use, no wiring or hardware altering was needed.

Fig 6. Turtlebot 2

Extended Battery

It is worth noting that the Turtlebot 2 did come with an extra extended battery to replace the existing smaller battery. This upgrade allows us the capability to run the Turtlebot a long time before the need of a recharge. Also, this battery is used to run not just the base of the Turtlebot, but also the Microsoft Kinect, and even the Arduino through a voltage regulator seetp. It is connected from a bottom compartment in the kobuki base.

Microsoft Kinect Sensor

The Kinect sensor is a neat addition to the Turtlebot's devices. It has a large capability due to its multiple sensors. Some data that it can provide include digital imaging, 3D pointcloud data, as well as range data. The range data is what we used to create an object detection program to allow for collision avoidance. It is firmly attached to the robot and has about a 60 degree view.

Kobuki Base

The kobuki base is what drives the Turtlebot. It uses the ROS system to receive its commands. The two wheels that are on the base are driven by independent motors. This is so that both motors can be driven in opposite directions or one faster than the other so the robot can turn.

Netbook

This computer came as part of the Turtlebot and is used as the terminal for all of "Max's" commands to be distributed to their proper locations. It runs our code and powers some of our modules. The Arduino's power comes from the netbook, which is then distributed to the GPS module as well as the magnetometer sensor. This netbook had

ROS libraries installed and is running the Linux Ubuntu operating system on it. Because we had little knowledge of the Ubuntu interface and operation, much time was spent in learning its ins and outs.

12.1.2 Adafruit Ultimate GPS Breakout

To narrow down our selection on how we were to implement a routing system and allow the robot to know where it is at all times, we decided to use GPS as it could be implemented outside where our project would be intended to operate. GPS was the best decision, as a visual mapping using the Kinect sensor of even a portion of the UTSA campus that would involve several buildings, would take a tremendous amount of computing power and time.

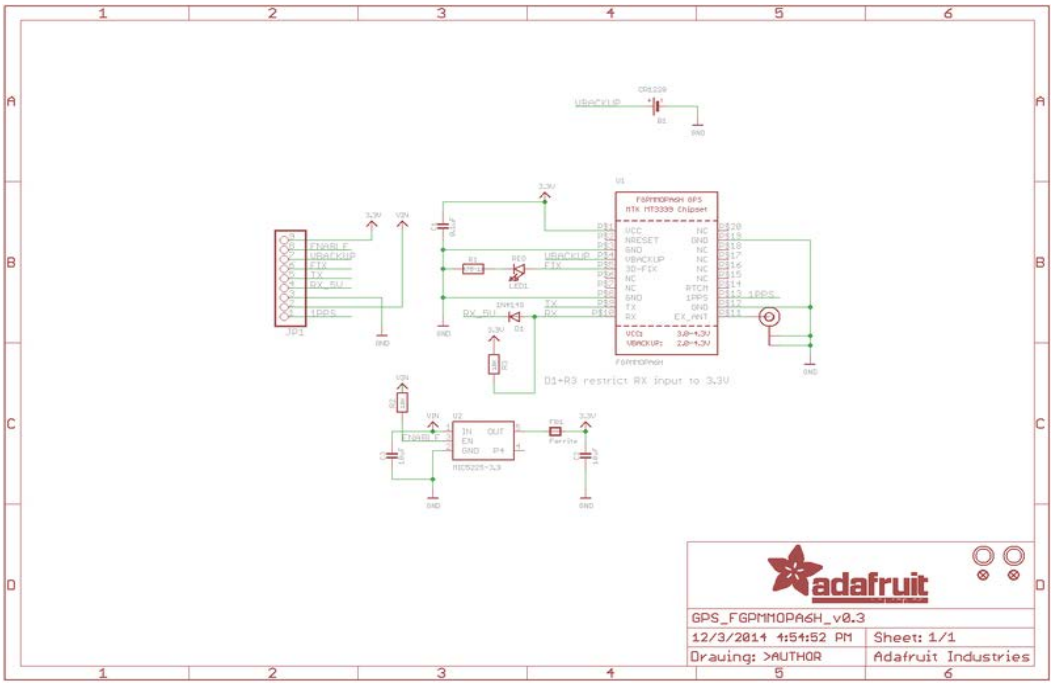

Fig 7. Schematic for the Adafruit Ultimate GPS Breakout [3]

The Adafruit Ultimate GPS Breakout is a GPS module that we chose to use due to its superior accuracy in GPS locating ability. The product specified an accuracy of less than 3 meters. The Adafruit module also allowed easy install of an optional antenna which we incorporated into our design. This specific package of the device is very compatible with our choice of microcontroller; the Arduino Mega 2560. The Adafruit Ultimate GPS Breakout provided us with the means in which to code very effectively with the Arduino Mega.

Fig 8. Antenna used with the Adafruit GPS

12.1.3 Magnetometer Sensor

With the use of GPS, or any locating and routing system, one always needs to know their heading; where we are pointing at the moment (respect to magnetic north). We first tried to start our robot facing north and calculate our heading do to our last movement. This however, brought problems in which we will discuss later. To combat this problem as well as to insure inaccuracies in the Turtlebot's motors don't tamper with our results, we decided to include a triple-axis compass magnetometer sensor.

Fig 9. View of the pins on the HMC588L Magnetometer Sensor

It is with this device we are capable of knowing our heading at any moment. We chose the HMC588L Magnetometer for several reasons. The accuracy on the device is advertised to be within three degrees, which is acceptable for the waypoint system in which we implemented. Also, the integration into the hardware system is very similar to the Adafruit GPS module. We can actually run this module alongside the GPS module as it uses the spare pins we had on the Arduino Mega.

12.1.4 Arduino Mega

In order to interface our locating and heading modules, we had to include a microcontroller. With so many to choose from, we decided on the Arduino Mega.

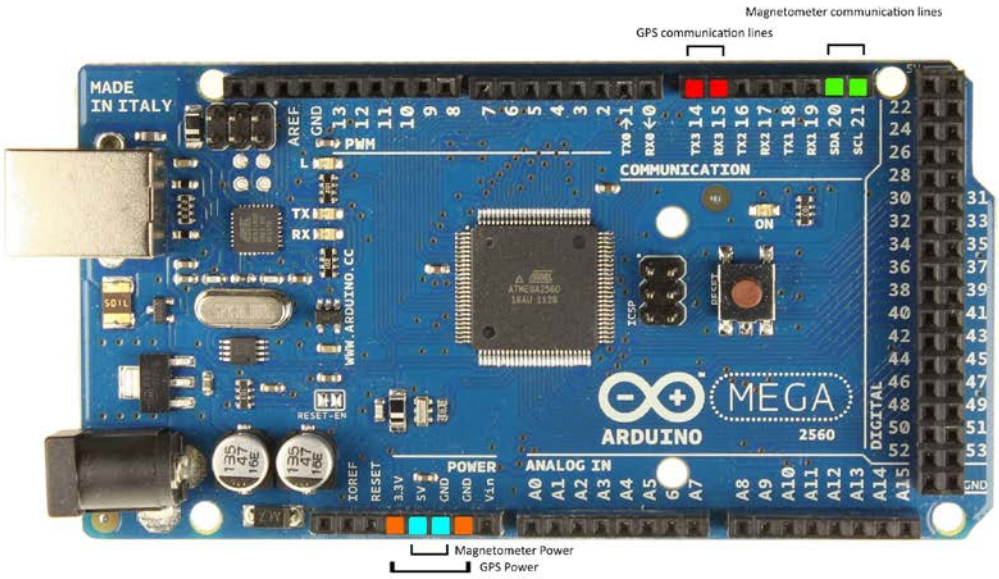

Fig 10. Arduino Mega 2560 Pinout

The Arduino Mega was the microcontroller we had at hand. It also suited our needs quite well in the fact it had multiple serial in/out pins. The reason we chose the Mega 2560 over the Uno was simply because of past experience with the Mega and also already having one at our disposal. Other than that fact we needed a little more processing power if we wanted to add other modules to the GPS later (such as a compass or a gyroscope) so we chose the Mega as it proved to be a potential problem solver in the future.

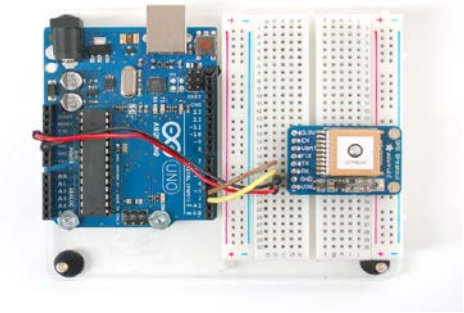

Fig 11. Wiring Guide for Arduino Uno

12.1.5 Wireless Router

All communication is done with the Robot Operating System, which needs a network, so a router was acquired to assign IP addresses and direct data travel. Setup time and complexity was not a factor with the wireless router as we had no need for internet access; just the IP address allocation. Because there were very few restrictions a cheap wireless router would have sufficed. This is why we chose the NETGEAR RangeMax WNR1000. With this particular device we gained the added benefit of having a slightly extended range for the network.

12.1.6 External Microphone

A microphone was fitted onto the robot in its final days of production. This addition was important as conflicts arose in testing. One of these conflicts included the microphone on the netbook being too close to the netbook's speaker. This resulted in the speech to text program responding to its own feedback. Also the microphone solves another issue with having to talk quite loudly to receive inputs, due to the location of the microphone on the netbook. With the netbook closed and very close to the ground, inputs from a standing person can be hard to retrieve.

12.1.7 Key Pad

A key pad was also added to the robot in the last few days of productions. We decided that having another form of input options for the user was best as we were having issues with the robot consistently hearing the user. This becomes very helpful for the robot when it is in situations where it would be too loud to hear the user.

12.2 Software

The hardware alone, of course cannot perform the tasks we intended for “Max”. The software portion is where the ingenuity and creativity is found and it is also the largest portion to this project. The software consists of hardware integration from the Arduino to the netbook, a speech-to-text program and its integration, and even the waypoint system in which “Max” uses to calculate his trajectories and even find his route. The different aspects of the software were split according to interest as well as coding background. The speech-to-text program and integration was assigned to Scott as he had some background with the Java computer language that the programs we used were written in. The ROS (Robot Operating System) commands, Arduino integration to the netbook, as well as the routing and waypoint system were entrusted in Cory. The Arduino IDE used and the code used in it were completed by Dylan.

12.2.1 The ROS System

Our multiple programs are intertwined and rely on each other’s information to supply other programs or devices with data and commands respectively. The ROS system provides a framework for communication between all of our programs. A large portion attributed to the preparation to our project design went into researching the ROS system.

Before the ROS system could even be used, some initial setup ensued. This setup included network installation on all computers that will be used in the control of “Max”. To give the ROS system a network of computers to talk to involve a router assigning IP addresses to each of the computers. With these address we assigned the netbook onboard the Turtlebot with a “master” tag which makes all nodes created, talk to it. A separate workstation computer was also assigned so that it can control the master and its nodes from a remote location. This is important because the netbook will be closed and placed in the Turtlebot under operation.

To begin each of “Max’s” sessions, a communication terminal must be initiated with “roscore”. What this does is set up connections in which all nodes can communicate to and from. Nodes are any separate executable file created from our programs and devices such as the Microsoft Kinect sensor and Kobuki base. ROS communications revolve around the sending of information, called publishing, and receiving information, called subscribing. Portions of each program are designed with ROS so the swapping of data is achieved. Each of these messages that are published/subscribed are done so to a topic in which any program can publish/subscribe to. Subscribers use callback functions to access this data and perform certain desired processes. Important functions of “Max” are done through ROS such as the velocity commands that run the Kobuki base’s motors at a specified velocity.

12.2.2 Speech-to-text Program

Speech-to-Text was one of our first ideas we wanted to implement in “Max”. It gave our project a kind of human feel which is not prevalent in most system like ours. It was also a parameter given to us by Professor Pranav.

Our Speech-to-Text program is the primary way that our uses will communicate with “Max”. Because of this fact we had to find one which had a large library of possible words as well as having the ability to incorporate it into the rest of our code. Our first step was to find the right library. We found that there are many different programs which convert speech to text, but the one that we chose was Sphinx4.

We found Sphinx4 while doing our research for the right program. Sphinx4 was being used by an open source code called VOCE. Although VOCE did have Speech-to-Text already in its programming it also had Speech Synthesis which was not necessary. Another problem with VOCE was that we couldn’t find a way to implement the output in

the rest of our code for “Max”. Because of these issues we decide to just use the Sphinx4 source code ourselves.

Once we got the Sphinx4 source code and implemented it we decided to take the output which was just being printed directly to the terminal and print it to a text file. Once we set up the code to print to the text file we found that every time the code convert our speech it would delete the previous text file and create a new one with the same exact name. This was ultimately helpful in our implementation.

Once we got the voice recognition working we decided that we want to have “Max” talk back to us. In order to make “Max” speak we decided to use “espeak”, a function found in Ubuntu which reads text files. In order to use espeak we had to create text files which contained the proper responses and questions to say to the user.

We decided the best way to implement our Voice-to-Text was to print the output to a text file as described earlier. Once the text file is created we have our code read the file and extract the string which was created by our Voice-to-Text. Once we extract the string, we set it equal to a string variable. Our next step then is to check that variable against all our possible location. If the string matches one of our possible locations then “Max” will begin its routing system to the user’s desired location. If the string does not match then the code will loop back to the start and try and get the user to say a place which “Max” recognizes.

12.2.3 Robot Control

The waypoint system and control is key to the functionality of the Turtlebot. It is where the user’s inputs are used to create a route from pre-determined waypoints in which the control system uses to traverse the UTSA campus. This program along with some other small subsidiary programs were created by Cory and encompass most operations of the robot and were almost entirely hand-made to handle each component such as the GPS, magnetometer, and Turtlebot itself. Below is a Software flowchart to

describe the process the control system takes in completing its goal, which is described in detail in the control program section.

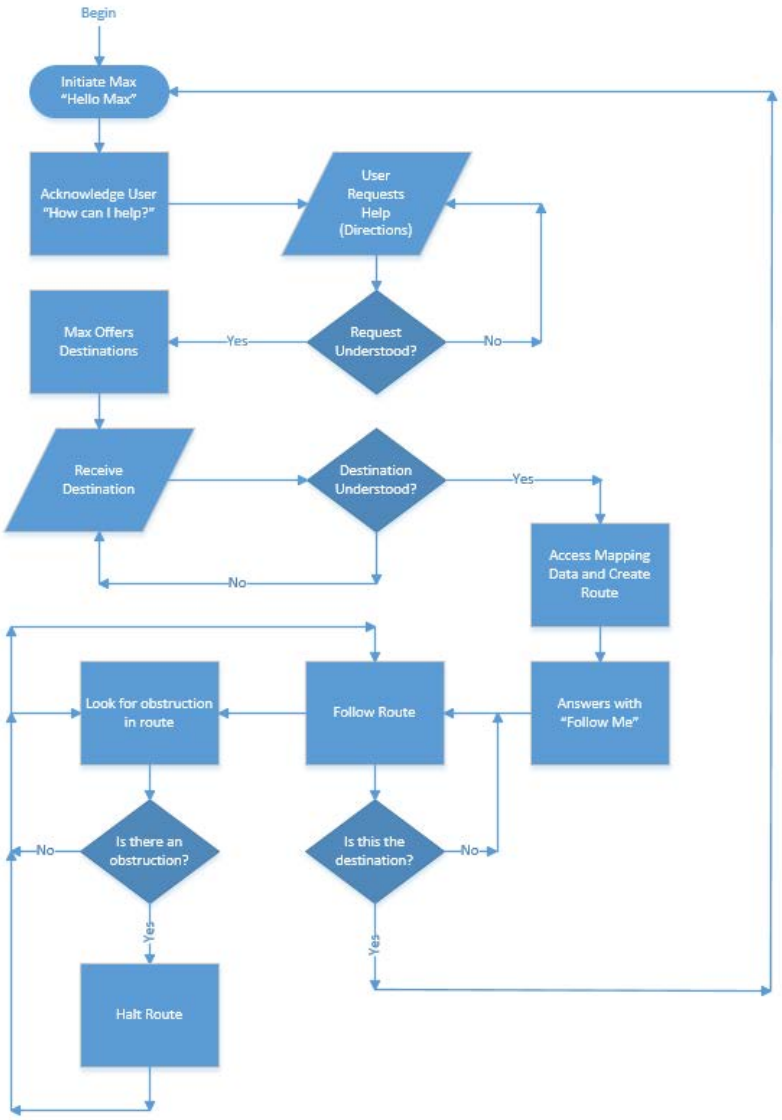

Fig 12. Software Flowchart

The system of programs consist of just two very important programs. One of which, the GPS/Heading program, is used to handle data integration from the Arduino and supply the much needed GPS and magnetometer data. This important program sends data through publishers in the ROS so that each node (Robot device or other program) can receive this information and handle it via device operation or further computation. All important data is captured and used at necessary moments within the second program, the Control program. With these two programs, the GPS data, Magnetometer data, Kinect range data, as well as Speech-to-Text program output are all incorporated correctly. A block diagram is depicted below to show visually the unification of the data captured and used by these programs.

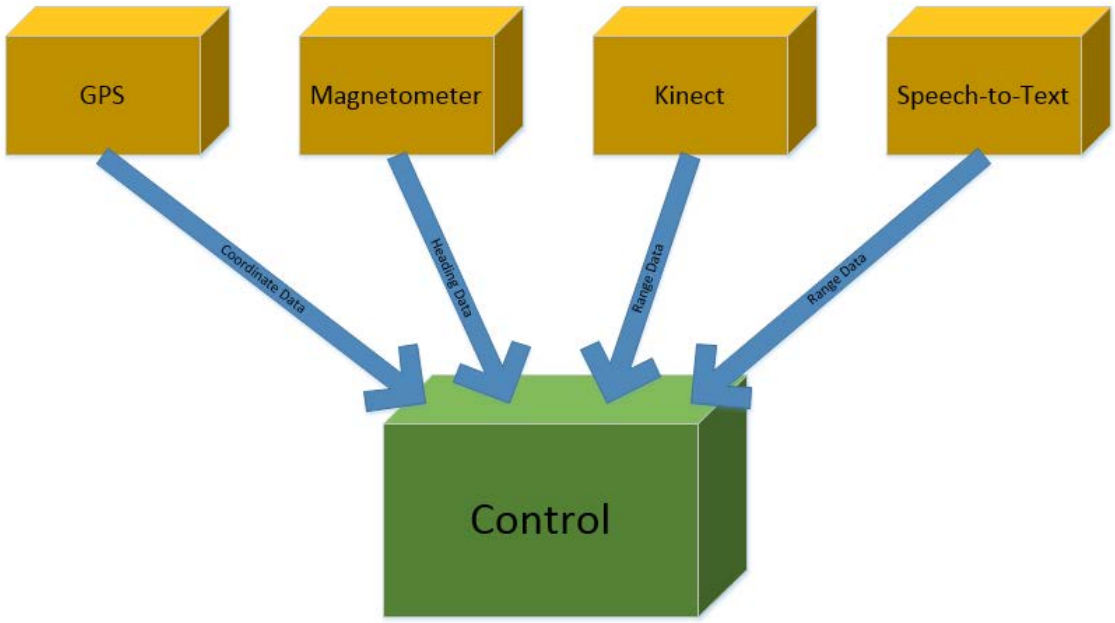

Fig 13. Program Unification Block Diagram

GPS/Magnetometer Program

The GPS program, called simply “gps_heading”, is a publishing program specifically intended for three purposes; to gather GPS coordinate data and magnetometer data by reading serial data from the Arduino, converting this data into usable variables, and then publishing this data for use by the control program. It is obvious that this program must run on the netbook as the GPS locations and magnetometer heading produced need to be of the robot’s location.

How this is achieved is by opening the USB port for communications with a specified baud rate. The baud rate that we are using to read the GPS data is 115200. [3] The next step is to receive the data. This is done with the read function on a loop that destructs when a line end or space is read. It however is not that simple. To effectively get all the data from the GPS module and magnetometer sensor, each bit must be read and stored into a buffer. After storing into a buffer, the buffer assigns its contents into an array of type char. All this is done while another variable increments to hold the point in which the array is being filled at.

Once the loop destructs, the first set of data is essentially read from the device. Now, the contents of the array which contain the values of the latitude and longitude and heading separated by commas is in a variable type that cannot allow us to do computation. It is here where we split the values of latitude, longitude, and heading into separate arrays and throw away the extra characters such as the spaces and commas used to split the latitude/longitude data and the next set of data. To do this, we used the strtok () function, which uses the commas and the terminating space to mark the separations points. Once the data is split, the data can be converted from the char type into a usable float number by using the atof () function. Finally, with numbers converted from characters read from the serial data at our disposal, we send it to the control program through the ROS system by publishing with separate messages on the “gpslat”, “gpslong”, and “heading” topics.

Control Program

The control program is the brain of “Max”, as it takes in all of the outputs from its subsidiary program, “gps_heading”, and uses them to control “Max’s” movements. The movement of “Max” is determined by a route created in this program. The route created uses hard-coded GPS coordinates and headings and routing based on start location and destination location.

To begin this large program, waypoints are set manually before running to have a static array of GPS coordinates that resemble points along a walkway of the UTSA campus. Multiple arrays are made for each different walkway. Strings are made reflecting all the possible destination points that “Max is capable of reaching.

Implementation of Speech to Text:

“Max” receives his destination from an output from the speech to text program. To acquire this output a simple procedure is done. The control program uses the system() function to run the java program that houses the speech to text and audio feedback. This java program outputs to a text file. After completion of the java program, a function is called that opens this output text file, reads the line that was saved to it, then closes the file.

If this information received matches a destination previously saved as a string, a destination point is established by setting variables “dest_lat” and “dest_long”. Here we take initial GPS data.

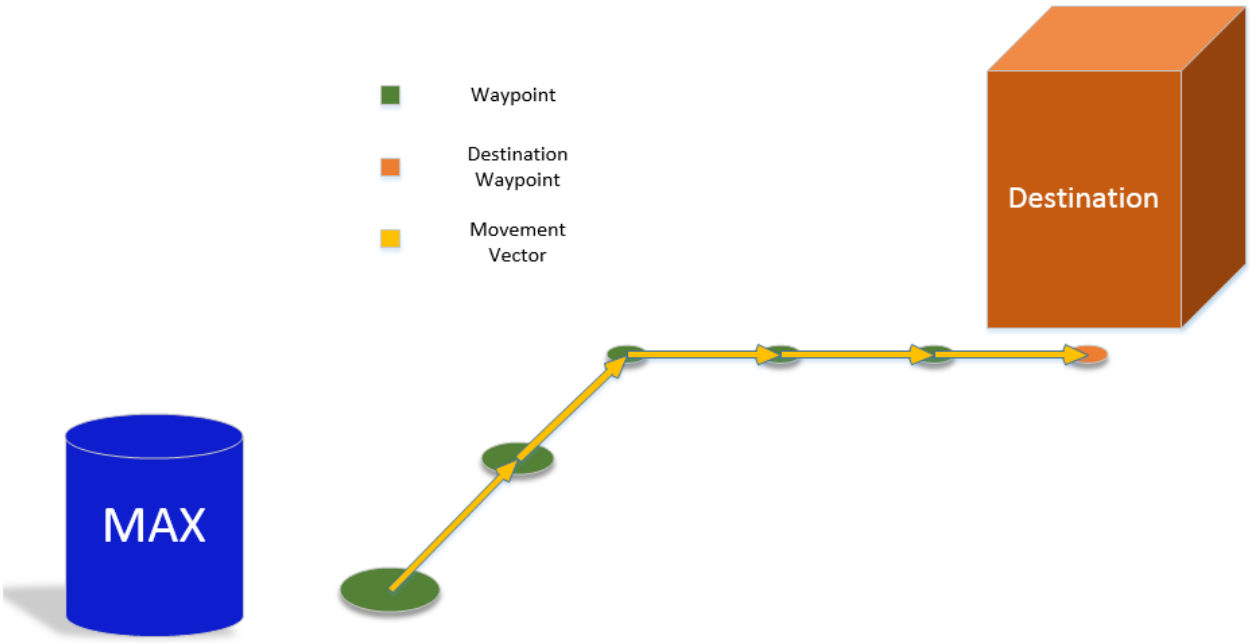

Fig 14. Visual Depiction of Waypoint Routing

With this initial setup, we jump into a loop if the current coordinates do not fall in a range of the destination coordinates defined by a variable “error”. Another loop is entered if the current coordinates do not fall in the range of the next waypoint coordinates. The loop inside of a loop structure is so that the process in the second loop can repeat if not yet at the destination. This gives a waypoint to waypoint route until the final waypoint is reached. How the second loop causes “Max” to move to the next waypoint is by simply applying the hardcoded heading and GPS data for that given waypoint. The initial approach of calculating the current vector by triangular calculations; proved ineffective due to GPS inaccuracies. With these values we publish a turn of the robot through radial velocity commands to the Kobuki base [5], while subscribing to the heading data published by the magnetometer program, until we meet the appropriate heading. After pointing towards the next waypoint, we move the robot forward by simply publishing velocity commands to the Kobuki base while subscribing to the GPS data so

that when we meet the appropriate range described earlier we stop the Turtlebot and break the second loop.

Kinect Range Data Implementation

However, we do not want to hit any obstacles in our path, so we subscribe to the data published by the Kinect program as we move forward, so a callback function is called upon periodically. The function is constructed of a for loop that calls for range data through a subscriber of topic “scan” for a specified amount of iterations to account for the full visual range of the Kinect sensor.. The range data is then individually checked to see if it is smaller than a set constant and then reiterated for the next array value. The constant acts as the distance we would like to stop the robot from an object. If an object is present within the range defined by the constant an object flag is set. When this object key is present (object = 1), the robot movement operation is halted until the object can no longer be detected and object key is no longer present (object = 0). If the path is clear, the process of moving is continued.

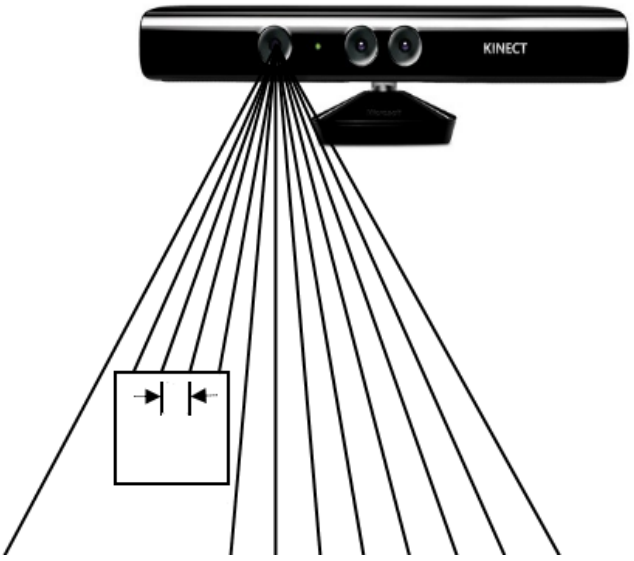

Fig 15. How the Kinect Sensor obtains range points

All this is done using the next waypoint coordinates and the current coordinates of the Turtlebot, which are periodically updated as described. This process is completed over and over until the first loop is no longer satisfied and the destination is reached.

12.2.4 Arduino Implementation

The first step with any Arduino related task is installing the IDE (integrated development environment) [2]. Installing it on the computer is simple and once installed to a PC, a library list must be added for the specific parts of the Arduino configuration you plan on having. [1] For M.A.D. “Max” the Adafruit GPS Breakout chip was our major component, and with the instruction manual comes with a link to the libraries for the GPS chip. [3] Once downloaded, they are moved into the directory of the Arduino operation folder and a whole new list of classes and subclass are available to use. Example code used for testing and gaining experience with the new library are also provided when the new library is installed.[2] The example code gave us the basic building blocks needed to go from no GPS data to concise and fairly accurate parsed GPS data. With basic structure and an understanding of how to set up the GPS in a basic way, the code was meticulous and chosen to be diverse and adaptive to the needs of the group, including call functions for many useful things such as current geological time, altitude, GPS data quality, date, and most importantly the GPS data.[3] Since we worked in a small area as compared to the entire globe, our GPS coordinate did not change at all in terms of degrees, as the UTSA Campus is on the same degree latitude and degree longitude no matter where you are on campus. So we decided to use the minute as our base 10 value so to speak. With minutes simply divide by 60 to get the seconds to get some useful data to use, as 1 second is about 90 feet. [4] Working with fractions of seconds, and minutes (all around campus adding up to .75 of a minute or so), was much easier than dealing with degrees, minutes, then seconds. This gave us the most useful way

of accessing our location and comparing the current data to destination locations and computing how to move there.

12.3 Engineering Analysis and Calculations

The design process began with the careful thought of each individual part. The project would be a mobile design and require battery power and will need to have a large enough run time to perform its duties. The GPS would have to be precise enough to give consistently accurate coordinates for our movement system. Also the waypoint system would have to account for the margin of error that the coordinates are capable of producing. The heading would also have to be accounted for as it also has a possible error range. It was put upon the group to account for these errors in calculation and use our finding to create a system in which can work with the inaccuracies.

12.3.1 Battery and Power Consumption

With the addition of the extended battery pack included in our Turtlebot 2 purchase our effective operating time of the Turtlebot improved from 3 to 7 hours. The netbook on the Turtlebot uses a 2500mAh Li-Ion battery. [20] The netbook itself uses a nominal value of around 500mA [20] while the GPS and magnetometer uses a nominal values of around 25mA [3] and 0.1mA [21] under operation respectively. The Arduino Mega's current draw is around 200mA. The calculations below show the power consumption of Arduino setup and approximate amount of time we are capable of running the netbook.

With the current draws from each device, we could add up the values to get a total current draw on the netbook's USB port:

$$500mA + 200mA + 25mA + 0.1mA = 725.1mA$$

With a total current draw we can find the approximate time we have to operate our netbook's before the battery is drained completely.

$$\frac{2500mAh}{725.1mA} = 3.4478h$$

Of course the netbook will not operate properly this whole time as it powers off automatically at 7% of its maximum battery state. Calculating this in, we have:

$$(100\% - 7\%) \times 3.4478h = 3.20645h$$

From these calculations the netbook would die rapidly and possibly not even provide the current necessary for the Arduino to function properly. Because of this, we incorporated a voltage regulator on the power pins tied into the power supply of the Turtlebot to power the Arduino.

12.3.2 GPS Coordinate Inaccuracy

To deal with the GPS inaccuracies, we had to set a range of acceptable points. However, the units for this range were unofficial as a degree of a coordinate on the North Pole is different than the value of a degree on the equator. For example, near the North Pole, a degree of latitude can be upwards of 70 miles while at that same point a degree of longitude can be only 55 miles. [22] Of course this is at the more extreme case, however it shows the ranges a degree of a coordinate can change. However, since we will operate our mobile directory in a rather small area respect to the globe, we had to worry less about this inaccuracy and more on the inaccuracy of the GPS unit itself. The advertised value is up to 3 meters. So to keep a safe range of distance, waypoints were taken to be at minimum of 6 meters away as the error for each the waypoint and the Turtlebot must be considered. However, while testing, the advertised accuracy of the GPS device was not proven to be true. This caused major problems as the Turtlebot did not have the means to traverse even the wider avenues of the UTSA campus. To help with this inaccuracy, we implemented a moving average filter to help keep steady data and WAAS. WAAS (Wide

Area Augmentation System) is a differential GPS system which uses highly regulated stations in North America to produce and transmit correction data to GPS satellites. We simply had to alter our setup in the Arduino code to enable this feature that came available on the GPS module we used.

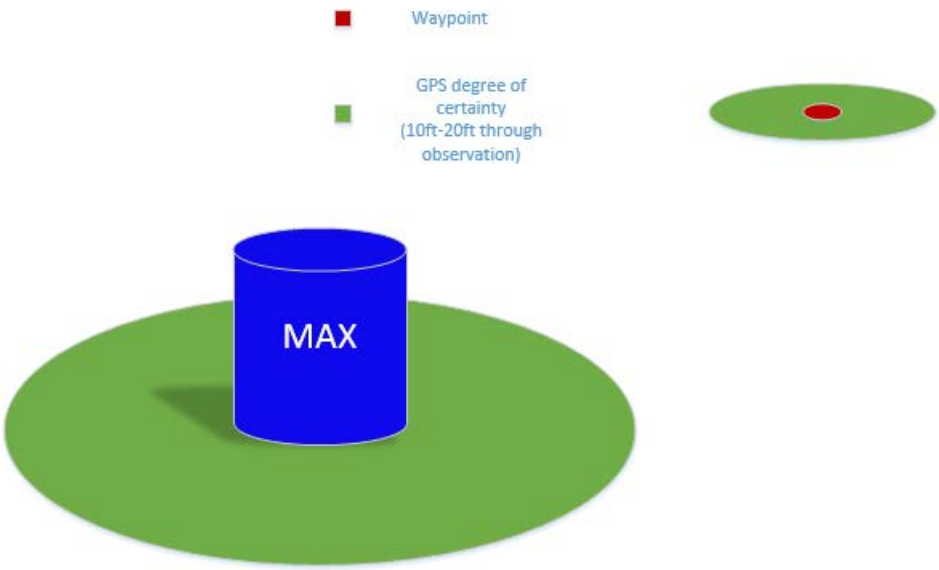

Fig 16. Depiction of GPS Inaccuracies

13.0 Major Problems

Throughout the development of the M.A.D. “Max” prototype the design team encountered minor problems. These problems were resolved in the most efficient, time saving, and cost effective manner possible. The design team’s ability to solve these major challenges show the efficiency and tenacity of the team and the value of each member’s individual efforts.

13.1 GPS/Magnetometer Integration

As comes with most projects involving modules in the design, our project consisted of a large amount of integration. The trouble arose when attempting to retrieve data from the

modules. Firstly, information retrieval through a coding environment had not yet been experienced by our team members. So to begin we scavenged the internet for help with the topic. However, for our particular dilemma, there were very few helpful resources. By combining ideas from multiple sources we were able to open and somewhat read data from the USB port. Once to that point, we found that the easiest way of taking in this data is to do it bit by bit and individually pulling characters from the serial output of the Arduino.

13.2 GPS Inaccuracy

As we had previously expected, our team encountered countless hours of trouble with the GPS inaccuracies. To combat this, we provided a range of acceptable GPS data. However, this still did not completely solve our problem as we had heading miscalculations due to the false GPS readings. As an attempt to create a working environment, we extended the range of each waypoint. One person in the group would literally pick up the Turtlebot and move it a desired length that we measured out and if the GPS data did not change enough we would extend the waypoint range further. What this did was make our heading data more accurate by giving a bigger difference in movement latitude and longitude, which in turn gave a more precise ratio to convert into a heading. The most improvement was due to the implementation of the WAAS system.

14.0 Integration and Implementation

As described in the major problems section, we had many issues in the integration of the hardware. To open a port for the USB and to read bit by bit was the solution. However, to see results we had to compare the data taken in and converted to the data actually being produced and set over serial. We ran the program several times as the serial monitor showed us the true GPS coordinates coming from the serial port. We would slightly modify the code each time to try and match the data. Finally, the data matched perfectly and would run for what seemed to be

indefinitely, without hiccup. After implementation of this GPS integration, we simply added similar functions to handle magnetometer heading data.

After integration of the modules, we moved on to the movement programs. To produce movement of the Kobuki base, we first had to learn the ROS system as it is the key. It was learned that the base takes a velocity command through publishing on the ROS system. Setup of a node and node handle had to be created in the initiation of the program controlling the base. We followed several tutorials and a published text to help guide us through the process of creating an ROS program. However, to run this program alone would tell us nothing. Luckily, the ROS world provides two different kinds of simulation; one more in depth than the other. For simple movement commands we used the turtlesim simulator. When the movement code was executed, the artificial 2D turtle would move across the screen as movement programs were publishing the velocity commands. However, it was noticed that the Turtlebot would not continue to move without a loop. After some research, it was found that the Kobuki base accepts inputs at a rate of at least 10Hz. So a loop was created around our publisher and its rate set to 10Hz, which was later upped to 20Hz. Finally, the turtlesim gave the result we were looking for; a steady stream of forward/turning movement commands.

Fig 17. Turtlesim

To integrate the Kinect Sensor into the programs we were again reliant on the ROS system. This time however, we had to subscribe to the Kinect instead of publish commands. To do so, we again referred to tutorials and text that reflected the operation we were after. However, to retrieve data from a Kinect sensor, we had to have one plugged in, which was not always an acceptable situation. This is where the other simulation software that ROS provides, comes in. This simulator is known as Gazebo and is a 3D simulation with virtual objects that can be placed. This helped greatly in the collision avoidance workings. Once we had an initial subscriber program, we had to edit intensively to gather range data from the Kinect sensor (or fake range data from Gazebo) and send it to the screen to visualize what the Kinect sees as range data. From this, we learned it was an array of ranges from all points of the Kinect's view. From here we designed a loop which checks for a range to close to the Kinect. If the range was to close a flag was published over ROS to the control program, effectively integrating the Kinect sensor. The flag, which is an integer value was passed correctly under the Gazebo simulation with a placed object in front of the Turtlebot. When ran on the actual hardware, the program performed as expected.

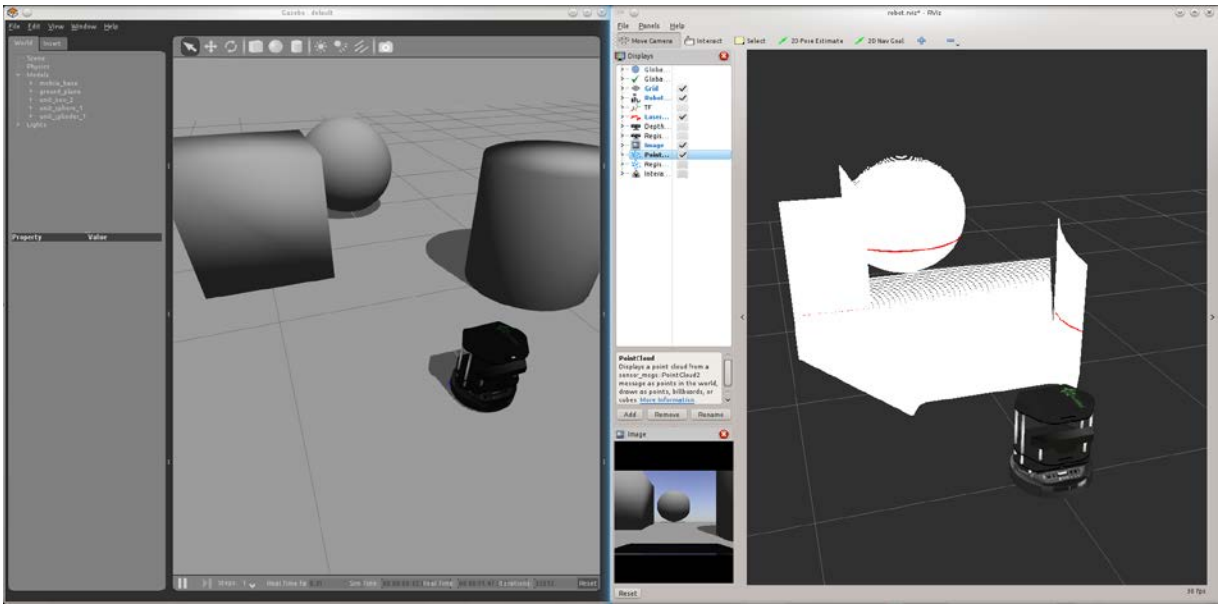

Fig 18. Gazebo and its Different Views

15.0 Comments and Conclusion

This design project has turned into something, we as a group, never would have expected. We enjoyed the experience of integration and the use of the GPS system and the speech to text program. Moreover, it was greatly appreciated that we had a sponsor in the process of our design. Dr. Pranav Bhounsule not only financially provided to our group, but also contributed his thoughts on our design in the biweekly meetings we had with him. He left freedom to the project in our hands and even allowed us the use of his lab.

The GPS waypoint system was truly a more complex system than had we anticipated. Because of the GPS inaccuracies, we found our success a heartfelt one. The integration of the system was a long drawn out battle in which we persisted.

The speech to text program gives a personal aspect to the project otherwise would not be possible. It was quite pleasing to see the Turtlebot have a sort of conversation with its user and actually do so in an effective manner.

A culmination of all the systems in action gave rise to a working prototype. With each separate aspect and each member's contribution, we were able to finish this project with a sense of accomplishment.

16.0 Team Members

The following sections refer to the individuals involved in the creation of the design project entailed in this paper. Each person contributed in their own aspects to the project, which was assigned to them due to their qualifications as an engineering student. It was a combined effort of these individuals that gave rise to our design prototype.

16.1 Curtis Dylan Odle

Dylan Odle, who goes by his middle name, is an electrical engineering student concentrating in controls. He is a Senior Leader with the UTSA Supplemental Instruction Program, with extensive collaboration and project management experience. As a former Computer Science major, Dylan holds knowledge in coding languages such as C, C++, HTML, and Java; as well as experience in software applications such as MatLab, PSpice, LabVIEW, and COMSOL. His experience in C and programming allowed for an initial understanding of Arduino Microcontrollers, which was used for the robot's GPS system, and provided him with enough skill to be up to the task of coding it. Dylan produced a GPS controller with extensive capability for the group's needs, as well as the robot's needs. Dylan was elected primary on the GPS system for his extensive background in coding in C and prior hardware experience. Dylan took responsibility for the GPS testing of the M.A.D. "Max" prototype. Dylan possesses an eye for seeing future problems, and made the executive decision to add the GPS Antenna himself. Dylan's execution and attention to detail made him vital to the design team's success.

16.2 Cory Royal

Cory is an Electrical Engineering student as the University of Texas at San Antonio looking to graduate in May of 2015. He is specializing in the controls concentration. Cory has worked with a program management company in the past in which he acquired a not only a project schedule mentality but also a large amount of troubleshooting skills. Cory has taken courses involving computer programming such as Digital Systems Design and Microcomputer Systems. His accomplishments in his school work and internship career will act as a testament to how he will conduct his business in this design project. With a decent background in C/C++, Cory took the lead in the integration and control aspects to the project. He also created the GPS waypoint system.

16.3 William Scott Morris

Scott will be graduating with a degree in Electrical Engineering in May 2015 from the University of Texas at San Antonio. Scott specialized in Controls concentration during his time at UTSA. During his time at UTSA Scott also took course which helped him throughout the entire Design II class which included Microcomputer Systems and Intro to Computer Programming. Scott also worked at BlackBerry during a summer internship which required him to design and implement his own code using JMP; a former of C++ and Java programming languages. This acquired skill allowed Scott to have a good understanding of using codes like C++ and Java which he used to implement the Voice-to-Text code in his Senior Design Project.

17.0 References

[1] Arduino.cc, 'Arduino - ArduinoBoardMega2560', 2015. Available:<http://arduino.cc/en/Main/arduinoBoardMega2560>

[2] “Language Reference.” Arduino – Homepage. Arduino. November 2012. Available:<http://www.arduino.cc/en/Reference/HomePage>

[3] Adafruit GPS Breakout Available:<https://learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf>

[4] Adafruit GPS Breakout Logger Shield Available:https://learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps-logger-shield.pdf.

[5] “A Gentle Intoduction to ROS”, Jason M. O'Kane, ISBN 978-14-92143-23-9, Available in PDF at: http://www.cse.sc.edu/~jokane/agitr/agitr-letter.pdf

[6] sourceforge.net, ‘VOCE’, 2015. [Online}. Available:<http://voce.sourceforge.net/api/c++/namespacevoce.html>

[7] sourceforge.net, ‘Sphinx-4 Application Programmer’s Guide’, 2015. [Online}. Available:http://cmusphinx.sourceforge.net/wiki/tutorialsphinx4

[8] The Institute of Electrical and Electronic Engineers (IEEE) Standards Association. Low-Rate Wireless Personal Area Networks (LR-WPANs). New York: IEEE, 2011

[9] The Institute of Electrical and Electronic Engineers (IEEE) Standards Association 1872-2015 - Approved Draft Standard for Ontologies for Robotics and Automation

[10] MacKinnon; Allan S. (Greene, NY), Willemsen; Donald J. (Ithaca, NY), Hamilton; David T. (Ithaca, NY), “Automated guided vehicle system,” Appl. No. 06/437,403, July 16, 1985

[11] Reeve; Peter J. (Warwickshire, GB2), Robins; Michael P. (Warwickshire, GB2), Roberts; Malcolm T. (West Midlands, GB2), “Vehicle control and guidance system,” Appl. No : 06/584,598, February 23, 1988

[12] Gudat; Adam J. (Edelstein, IL), Bradbury; Walter J. (Peoria, IL), Christensen; Dana A. (Peoria, IL), Kemner; Carl A. (Peoria Heights, IL), Koehrsen; Craig L. (Peoria, IL), Kyrtsos; Christos T. (Peoria, IL), Lay; Norman K. (Peoria, IL), Peterson; Joel L. (Peoria, IL), Schmidt; Larry E.(Peoria, IL), Stafford; Darrell E. (Dunlap, IL), Weinbeck; Louis J. (Peoria, IL), Devier; Lonnie J. (Pittsburgh, PA), Rao; Prithvi N. (Pittsburgh, PA), Shaffer; Gary K. (Butler, PA), Shi; Wenfan (Pittsburgh, PA), Shin; Dong Hun (Pittsburgh, PA), Sennott; James W. (Bloomington, IL), Whittaker; William L. (Pittsburgh, PA), West; Jay H. (Pittsburgh, PA), Kleimenhagen; Karl W. (Peoria, IL), Clow; Richard G. (Phoenix, AZ), Wu; Baoxin(Pittsburgh, PA), Singh; Sanjiv J. (Pittsburgh, PA), “System and a method for enabling a vehicle to track a preset path,” 5610815, November 17, 1998

[13] Bruemmer; David J. (Idaho Falls, ID), Few; Douglas A. (Idaho Falls, ID), Walton; Miles C. (Idaho Falls, ID), “Robotics virtual rail system and method,” US 20080009964 A1, Jan 10, 2008

[14] Takafumi Sonoura, Kaoru Suzuki, “Interactive robot, speech recognition method and computer program product”, US7680667 B2, Mar 16, 2010

[15] Ki Beom Kim, “Speech recognition method for robot “, US20120130716 A1, May 24, 2012

[16] Xavier Anguera Miro, Roland Kuhn, Luca Brayda, “Interactive personalized robot for home use”, US7349758 B2, Mar 25, 2008

[17] ROS Wiki, 'ROS Wiki, Turtlebot', 2015.Available:http://wiki.ros.org/Robots/TurtleBot

[18] ROS Wiki Tutorials, 'ROS Wiki Tutorials, 2015. Available:http://wiki.ros.org/ROS/Tutorials

[19] ROS.org, 'Turtlebot Simulator', 2015. [Online]. Available:http://wiki.ros.org/turtlebot_simulator

[20] “Travelmate B”, TMB113-M-6826, us.acer.com/ac/en/US/content/professional-model/NX.V7QAA.018

[21] “3-Axis Digital Compass IC HMC5883L”, Honeywell; http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5883L_3-Axis_Digital_Compass_IC.pdf

[22] “USGS FAQs”: “How much distance does a degree, minute and second cover on your maps?” http://www.usgs.gov/faq/categories/9794/3022

[23] “Getting the Most from GPS”, Wayne’s Tinkering Page. https://sites.google.com/site/wayneholder/self-driving-rc-car/getting-the-most-from-gps