THE UNIVERSITY OF TEXAS AT SAN ANTONIO

SHUSHBot

Smart Humanoid Used for Silencing Humans Robot

4/24/2015

Table of Contents `

Table of Figures

1.0 Executive Summary

John Peace Library (JPL) has commissioned the use of a Human-Robot Interactive machine (SHUSHBot), shown in figure 1, for two main purposes; the first is the lack of staff, which has led to higher noise levels within the JPL. Certain areas of the JPL are designated as "Quite zones." However, these zones are limited in seating and cannot be monitored at all times. With this problem at hand, Stacy Alexander, Thilo Janssen, Roberto Mexquitic, and Javier Gonzalez were given the opportunity to design and develop a human interactive mobile robot. Today the SHUSHBot team is currently debugging and updating the active prototype as planned in the prior semester. Currently the mobile robot reacts to high noise levels and warns the human of his/her being too noisy.

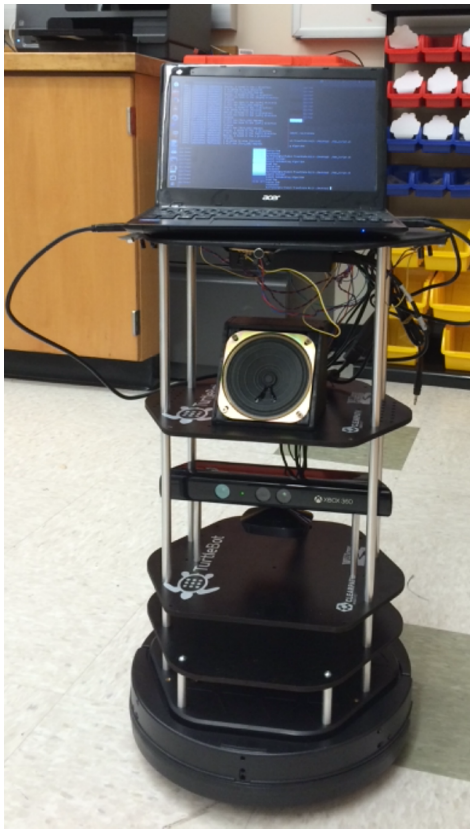


Figure 1. SHUSHBot

The second goal is to serve as an attraction for guests visiting the JPL and as a recruiting tool for the university and the college of engineering. It was assessed that SHUSHBot would use the microphone within the laptop to detect and decrypt speech patterns, such that a student may greet "Hello" and SHUSHBot would reply with a simple "Hello" as well. A deviation was made this semester due to the primary computer having a malfunction, thus needing us to use a different computer that has no microphone installed within it. To solve this problem we used a

USB connected microphone to attach to the laptop. The Blu microphone is temporarily used while we find an alternative lighter and smaller microphone that will allow speech interpretation.

2.0 Introduction

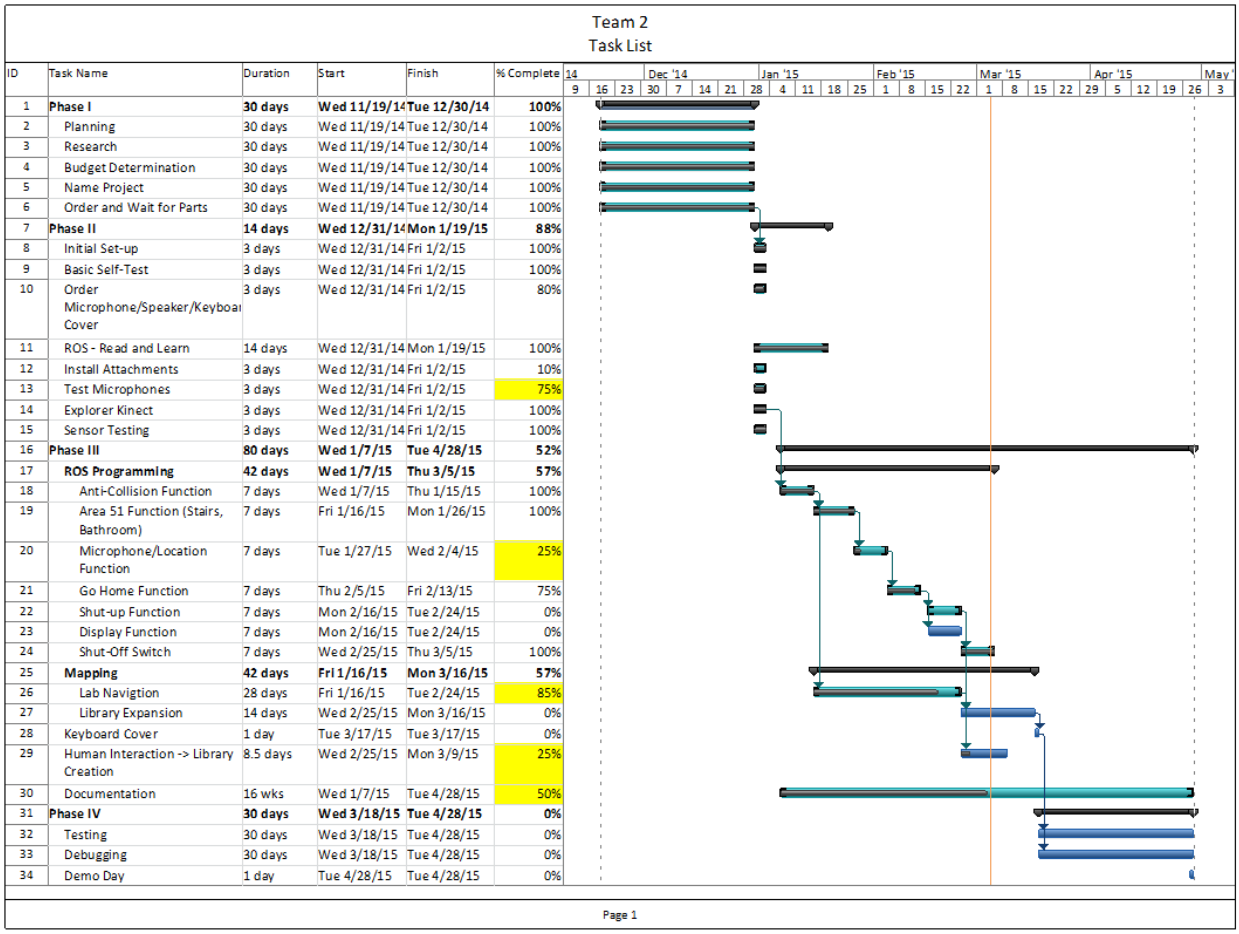
Disney Land/World has human interactive robots which catch the attention of young children. The interaction between the younger people and the robots cause an increase of attendance to the theme park. These robots in the theme park provide directions to those who are lost, interact with people via “Hello, Goodbye, and how are you,” and more advanced robots move even through the park communicating with the guests.

The project, the “Smart Humanoid Used for Silencing Humans Bot” (SHUSH Bot), will have a primary task of navigating the 3rd floor of the JPL while monitoring the noise level depending on the position of the bot. As a secondary function it might allure new students to apply to the university as well as draw more students into engineering.

In our research to create an adequate and inexpensive robot as a platform, we discovered the TurtleBot 2.0. The TurtleBot is a small, compact mobile robot which uses a Roomba as a basic frame. It comes prebuilt with a computer, various outputs, a 360 Kinect camera, and platform to install additional components. Through our research analysis we found that the TurtleBot software includes a Software Developers Kit (SDK), libraries for visualization, planning, and perception, control and error handling, and a development environment for the desktop. With a low cost and ready to play out of the box we anticipated a significant reduction in construction time.

We applied a Pugh Matrix in order to optimize our design and parts, and to outline our budget requirements. An alternative design idea was to add a robotic arm to the bot giving it the ability to wave when greeted and point in the direction of a certain area when asked for it. This would greatly increase the robots interaction function as well as its appeal to the public. Though, this design was rejected because of time and budget constraints.

In figure 2 our fully developed project schedule can be seen. The team has set an end date one month before the presentation day. This will apportion enough time to debug and hopefully add additional functionality to SHUSHBot so it may perform at its peak. By March 18, 2015 the team has completed assembly of the fully operational SHUSHBot, which is also compatible with any user. From there on our emphasis will be on debugging and fine tuning the system. Our main design will focus on navigating the JPL and checking the noise level. For this function we require an array of noise sensors to pick up noise and locate its origin. The final decision was incorporated into a function block diagram, as shown by figure 5, outlining various components of the design. Afterwards, a software function block diagram, illustrated by figure 6, was created illustrating the software flow of the design.



ID	Task Name	Duration	Start	Finish	% Complete
1	**Phase I**	**30 days**	**Wed 11/19/14**	**Tue 12/30/14**	**100%**
2	Planning	30 days	Wed 11/19/14	Tue 12/30/14	100%
3	Research	30 days	Wed 11/19/14	Tue 12/30/14	100%
4	Budget Determination	30 days	Wed 11/19/14	Tue 12/30/14	100%
5	Name Project	30 days	Wed 11/19/14	Tue 12/30/14	100%
6	Order and Wait for Parts	30 days	Wed 11/19/14	Tue 12/30/14	100%
7	**Phase II**	**14 days**	**Wed 12/31/14**	**Mon 1/19/15**	**88%**
8	Initial Set-up	3 days	Wed 12/31/14	Fri 1/2/15	100%
9	Basic Self-Test	3 days	Wed 12/31/14	Fri 1/2/15	100%
10	Order Microphone/Speaker/Keyboa Cover	3 days	Wed 12/31/14	Fri 1/2/15	80%
11	ROS - Read and Learn	14 days	Wed 12/31/14	Mon 1/19/15	100%
12	Install Attachments	3 days	Wed 12/31/14	Fri 1/2/15	10%
13	Test Microphones	3 days	Wed 12/31/14	Fri 1/2/15	75%
14	Explorer Kinect	3 days	Wed 12/31/14	Fri 1/2/15	100%
15	Sensor Testing	3 days	Wed 12/31/14	Fri 1/2/15	100%
16	**Phase III**	**80 days**	**Wed 1/7/15**	**Tue 4/28/15**	**52%**
17	**ROS Programming**	**42 days**	**Wed 1/7/15**	**Thu 3/5/15**	**57%**
18	Anti-Collision Function	7 days	Wed 1/7/15	Thu 1/15/15	100%
19	Area 51 Function (Stairs, Bathroom)	7 days	Fri 1/16/15	Mon 1/26/15	100%
20	Microphone/Location Function	7 days	Tue 1/27/15	Wed 2/4/15	25%
21	Go Home Function	7 days	Thu 2/5/15	Fri 2/13/15	75%
22	Shut-up Function	7 days	Mon 2/16/15	Tue 2/24/15	0%
23	Display Function	7 days	Mon 2/16/15	Tue 2/24/15	0%
24	Shut-Off Switch	7 days	Wed 2/25/15	Thu 3/5/15	100%
25	**Mapping**	**42 days**	**Fri 1/16/15**	**Mon 3/16/15**	**57%**
26	Lab Navigtion	28 days	Fri 1/16/15	Tue 2/24/15	85%
27	Library Expansion	14 days	Wed 2/25/15	Mon 3/16/15	0%
28	Keyboard Cover	1 day	Tue 3/17/15	Tue 3/17/15	0%
29	Human Interaction -> Library Creation	8.5 days	Wed 2/25/15	Mon 3/9/15	25%
30	Documentation	16 wks	Wed 1/7/15	Tue 4/28/15	50%
31	**Phase IV**	**30 days**	**Wed 3/18/15**	**Tue 4/28/15**	**0%**
32	Testing	30 days	Wed 3/18/15	Tue 4/28/15	0%
33	Debugging	30 days	Wed 3/18/15	Tue 4/28/15	0%
34	Demo Day	1 day	Tue 4/28/15	Tue 4/28/15	0%

Figure 2. SHUSHBot Project Schedule

Team 2 is composed of 4 students that cover many sub-fields of electrical engineering. These include Digital Signal Processing, Systems Controls, Computer Engineering, and Power Engineering. Several team members have a background in computer programming, research, internship, and prior work experience. Team 2 members divided the work need to be done according to the schedule created.

3.0 Need Being Addressed

Our main objective is to reduce distracting noises within the library and attract students to the university. SHUSHBot should operate autonomously with a limited degree of freedom to explore its environment. It needs to gather information about the environment, safely move around without harming any people, property, or itself. Fulfilling the task to work for an extended period without human intervention requires several programming specifications. These will involve several objectives including navigation through the library, obstacle avoidance and

noise detection algorithms. We will have to address the issue of battery life operation and mapping of the JPL. A detailed examination of each component will be required for a successful project outcome. In figure 3 we illustrate the concept of SHUSHBot. There will be a quiet zone, in which SHUSHBot tell people to lower their noise levels. There is a home area in which SHUSHBot will go to after low battery alert is displayed and attend to its charging station. There is also an interaction phase in which SHUSHBot will be outside the quiet zone and interact with any passer that comes by.

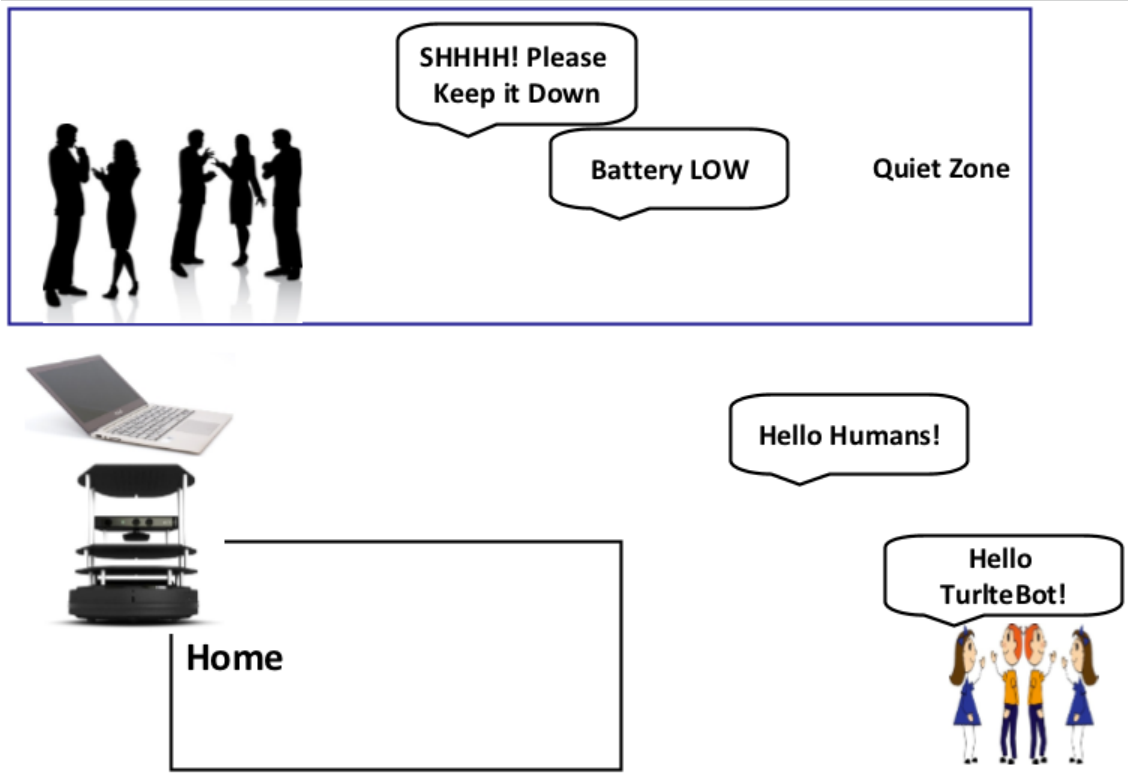


Figure 3. The Concept of SHUSHBot

4.0 Literature and Patent Search Results

Human to robot interaction has become a new notion within the robotics community, and there are also companies in which wish to develop newer technologies to help develop these new notions. Along with designing and developing new human interaction technologies there are certain constraints, such as patents, we as engineers must abide by. Although our patent search had relevant articles to SHUSHBot, all of them were not relevant in the design and development of SHUSHBot. To start off the design we needed a method of movement based on sound detection. To avoid legal issues we preformed a Google Patents search, in search of any company of research lab that had developed such device. In our search we came across an interactive robot in which decodes speech and is able to make an estimate of where the speech comes from.

5.0 Marketing Analysis and Market Strategy

SHUSHBot will solve the high noise level problem within located quiet zones. Disney Land/World, Six Flags, Sea World and any other theme parks that can involve interaction of a machine and a human. They might pay for such device because human to robot interaction is not widely known; it can mainly be seen within research labs. Because it is not so common, this human to robot interaction will be a new experience to people within theme parks and general populated areas such that people will enjoy as well. In this case the purpose of SHUSHBot would not only be to silence those in the theme park but also would have a functionality of interacting with people.

Populated theme parks and/or libraries would be interested in making human interaction robots mainly for the experience of the consumer. The purpose of SHUSHBot is to silence loud students in the library of the university and create a comfortable experience between the student and SHUSHBot. Therefore, this robot would be mainly used for research and not for commercialization.

6.0 Engineering Design Constraints

Global Constraints

1. **Engineering Codes and Standards:**
   The device will need to have standardized battery power
2. **Economic Factors:**
   The final cost of the product must be affordable and should be simple to acquire.
3. **Environmental Effects:**
   The batteries are rechargeable batteries.
4. **Sustainability**
   The product can be sustained over long term as long as the software and maps are updated as needed.
5. **Manufacturability (Constructability)**
   The product should be manufactured in the USA and software stay in the company.
6. **Ethical Consideration**
   The Kinect should not be activated anywhere near a restroom or where video recording is not allowed.
7. **Health and Safety Issues**
   The robot needs to have adequate obstacle avoidance. The robot needs to have an emergency off switch in case of malfunction.

8. **Social Ramifications**
   The robot is not replacing a human's job, but assisting in the job of the librarian.
9. **Political Factors**
   Currently this is not an issue. In the future if robots start replacing the jobs of humans the use of robots may become a key political factor.
10. **Legal Issues**
    The use of the product should not interfere on other patents, since nothing comparable is on the market.

Local Design Constraints

1. **Cost**
   Development costs of the product are set by the product sponsor.
2. **Schedule**
   The deadline for the development of the product is by the end of spring 2014 semester, the end of Design II class.
3. **Manufacturability**
   The equipment is a combination of readily available parts and amplifier applications. These applications have to be designed on PCBs and reliable produced.
4. **Engineering Codes and Standards**
   The robot needs to have an emergency shut off in case of malfunction. All batteries and other hardware components must comply with existing codes and standards.
5. **Ethical Considerations**
   The robot needs to be kept away from sensitive data, restrooms, or anywhere a camera is not permissible.
6. **Health and Safety Issue**
   The robot needs to have adequate obstacle avoidance to avoid injuring bystanders. The robot needs an emergency shut off switch in case of malfunction.
7. **Legal**
   This does not apply, since employees should not be affected by safety issues in the manufacturing process.
8. **User Requirements**
   A manual needs to be written with all associated warnings and clear and concise language.

7.0 Product Requirements

The SHUSHBOT project's product requirements and specifications transformed slightly over the course of the project. The facial recognition and adaptive mapping product specifications were no longer required as the scope of the project was reduced so the requirements were obtainable with limited time. Specifications including the budget, safety, sound recognition, and human interaction became more defined as the goals we strived for.

- Battery Life
- Static Avoidance
- Adaptive Mapping
- Sound Recognition
- Dynamic
- Facial Recognition
- Human Interaction
- Kill Switch
- Power Management
- ROS/OOS Integration
- ANSI Standards/Safety
- Complexity/Time Available
- Upgradeable
- Cost

8.0 Engineering Codes and Standards

The IEEE organization was founded in 1884 at a time where electricity was becoming accessible to an increasing number of people. [4] Standards were created for both compatibility of products and safety of consumers. Many of these standards have been used in the SHUSHBOT project; some examples of IEEE standards are listed below.

- **IEEE std 1625 -2008** - Standard for Rechargeable Batteries for Multi-cell Mobile Computing Devices:

  The IEEE 1625 standard provides guidelines for testing, designing, evaluating, and interfaces between subsystems. This particular standard also includes end-user awareness to maximize the reliability of a system [1].

- **IEEE std 802.11 - 8802-11-2012** - Information technology -Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications

  This standard is the updated version of the 802.11 incorporating the original amendments from 2008 – 2010. It sets standards for wireless local area networks. [2]

- **IEEE std 1680.1-2009** - Standard for Environmental Assessment of Personal Computer Products, Including Notebook Personal Computers, Desktop Personal Computers, and Personal Computer Displays

This standard deals with the issue of how to reduce the environmental impact of electronic products, specifically computers and displays. [3]

9.0 Design Concepts

The current design idea for Group 2 is to design a robot to roam around the JPL's quiet zones reminding students to be quiet thereby respecting the other students working around them. The robot's base design will be a TurtleBot which we will add a display panel that will act as a face and provide non-verbal communication. The robot will also contain an array of sensors to determine the location of the person making noise exceeding our set threshold. There will also be a speaker to help enhance the human interaction component desired by our client.

Pugh Matrix				
Design Constraint/Requirement	**Weight**	**Design 1**	**Design 2**	**Design 3**
Battery Life	10	4	6	7
Static Avoidance	4	5	6	6

Adaptive Mapping	5	10	10	10
Sound Recognition	6	9	6	6
Dynamic	5	4	6	6
Facial Recognition	4	7	8	8
Human Interaction	13	10	9	4
Kill Switch	5	4	6	7
Power Management	10	3	6	7
ROS/OOS Integration	8	6	7	8
ANSI Standards/Safety	8	4	5	6
Complexity /Time Available	10	2	6	8
Upgradeable	3	3	5	7
Cost	9	3	6	7
	100	**528**	**638**	**688**

Figure 4. Pugh Matrix

We have defined three different design approaches for our interactive robot. The first design we called *All Attachments*, this is the most ideal design to meet the human interaction requirement. It has the most sensors, a speaker, a monitor acting as a face, and an external arm but is also the most complex and time consuming of the three design concepts. The second design still contributes to human interaction but does not include an arm and has a decreased number of sensors for determining sound location. This design concept will give us a longer battery life and has a lower cost than the *All Attachments* design. The third design concept has the lowest cost of all three designs, the least time and be the least complex. This design does not include an arm, less sensors to determine location of sound source, and does not have a speaker. It is the most upgradable and has the best battery life and still meets minimal human interaction criteria.

The first design constraint we considered was battery life. If our robot does not have a sufficient battery life of operation then it is inefficient to deploy. As well as being able to operate with various modules and peripheral that it must power as well. This also goes along with power management, the ability to monitor and control the amount of power needed by not only the robot itself but any additional accessories we add. Static Mapping is referring to the ability of our robot's ability to map a room and navigate through it. Our main goal is to have human

interaction first then have the robot maneuver around a fixed area. Because the Library is not a static area, with moving chairs, backpacks, furniture, and other objects our robot should be able to adaptive to new environments which is where adaptive mapping is a goal. This will be one of the most complex parts and will likely be the last be the part of our project. Our robot should be able to tell when a set sound threshold has been crossed and execute a certain command from it. This will be important for attaining the goal of keeping people quite in the library. The weighing for the Robot Operating System/Object Oriented Software was established at a medium "8", since the ROS is the open source software delivered with the TurtleBot and plays a major role in the component integration. The ANSI/RIA R15.06-1999 American National Standard for Industrial Robots and Robot Systems is set to a medium scaling, since we do not work with a high powered strong robot, but should follow all given standards to a newly build robot, e.g. rule 4.6 "every robot shall have stopping functions providing for one or more emergency stop devices." We also have included a goal of adding an external kill switch that would be highly visible and an easy way for anyone to turn off the robot in case of an emergency. The dynamic avoidance is set to a lower weight due to slow walking personnel in the library and no expectations of sudden erratic motions. The TurtleBot needs to have a soft- and hardware design, which is upgradable for future design students. However, this is not a large concern of our team. Facial Recognition refers to the SDK Kinect camera will be able to determine different facial expressions (Smile, Sad, Mad, and Cry) to be able to interact with the person. The human interaction is to make the robot more likeable and human like to attract attention, for the purpose of the design, the robot will go around the library to "shush" people that are above threshold noise. Cost and complexity are primary concerns, even though this project is funded by the professor, we are still trying to budget low and efficient, in case we decide to add upgrades.

In conclusion, design 1 has the lowest rating and appears to be the most difficult to attain given the time and financial constraints we face. Although design three received the highest rating our goal is still to aim for design two, the addition of a speaker will make a significant positive impact on the human interaction component of our project.

10.0 High level Block Diagrams

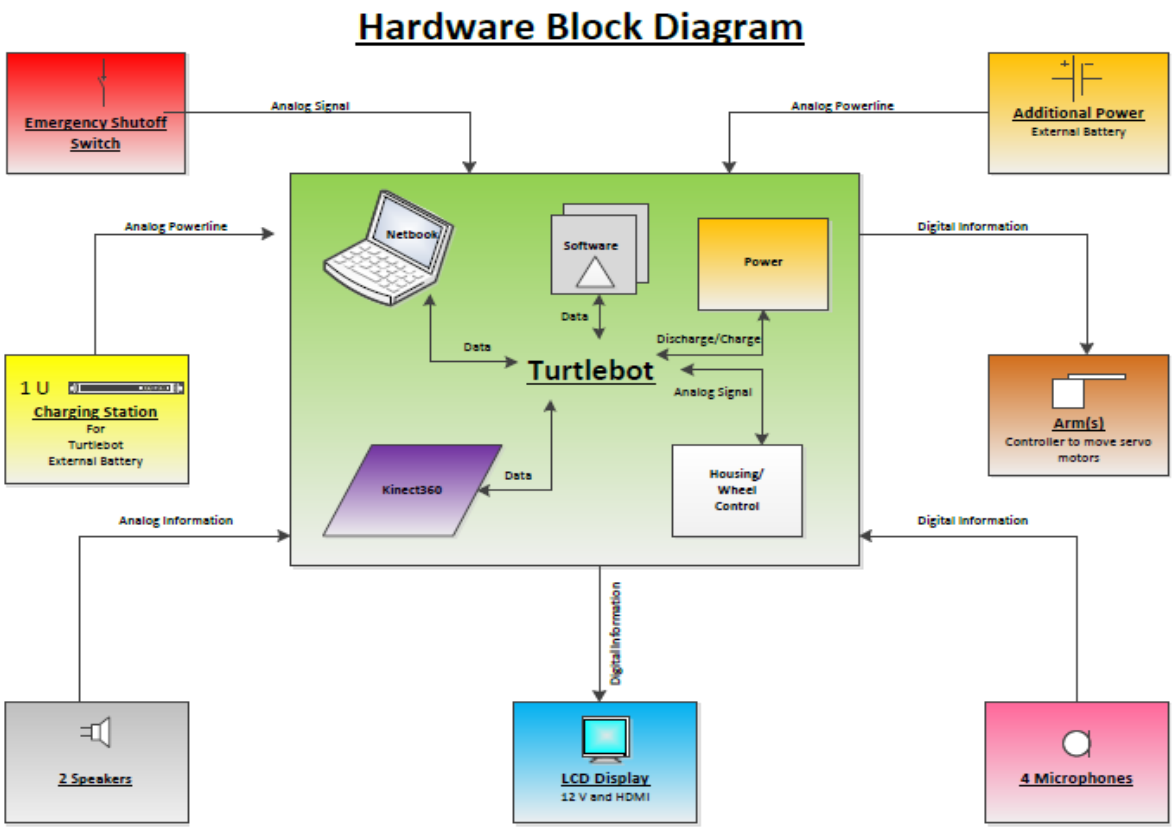


Figure 5. Hardware Functional Block Diagram

Software

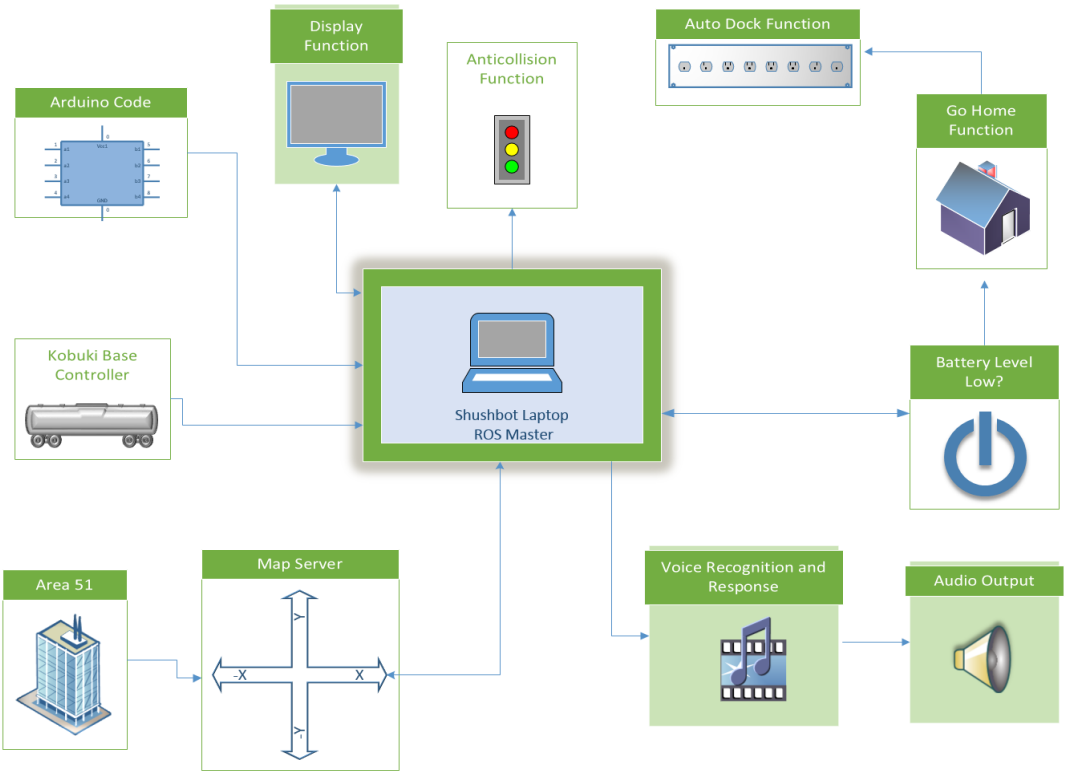


Figure 6. Sofware Function Block Diagram

11.0 Major Components

The major components on SHUSHBot are the sensors, the Kobuki, and the Robotics Operating System. Primarily we needed an inexpensive efficient mobile robot tat can maneuver around obstacles decently. In choosing Turtle bot to be the base of SHUSHBot we looked at size efficiency and what came in the package, such that what came with the bot. It came down that no matter what mobile robot we chose, Turtlebot was the cheapest robot that included many things, and also used in research.

The sound sensors we also important, we searched for high quality sound sensing electronics that had variable outputting voltage signals. Even though we made slight modifications to the sound sensors, the sensors bought were at a higher sensitivity that an inexpensive sound sensor. The measures taken in choosing the specific sound sensors were not considered at all.

The company who built the Turtlebot acquired the operating system for SHUSHBot. We had no other choice for SHUSHBot, but it is a major component to SHUSHBot.

12.0 Detailed Design

Hardware

The hardware development was incorporated according priorities. The Turtlebot II (inner diagram figure 5) was delivered with a functional Kinect 360, Housing and basic Wheel control (Kobuki), a regular rechargeable battery, a Netbook with a battery/charger and a basic software package. These hardware accessories did not need to be modified. External hardware (outer part of figure 5) had to be purchased and modified to meet the required specifications. The team prioritized on various devices to follow a tight time schedule. The charging station was purchased with the initial order of the Turtlebot in order to charge the basic battery. The team decided to use the LCD Display of the netbook for budgetary reasons. An additional battery was also order with the first order, but was not incorporated into the power scheme. Four microphones were ordered after the delivery of the Turtlebot, with the focus of incorporating them as soon as possible. While waiting for the delivery, an external speaker with its amplifier was developed. Additionally, the team decided to increase the height of the Turtlebot by eight inches to facilitate the human interaction part, which caused a renaming to SHUSHBot. The emergency shutoff switch had the lowest priority of our group and was intended to be integrated last with the Plexiglas cover for the notebook.

Kinect 360, Kobuki, Battery, and Netbook:

The Turtlebot II was delivered from the manufacture Clearpath of Canada with the basic construction. The base contains the Kobuki with a battery compartment, various power supplies, charging contacts, power switch, housing, and wheel control. A detailed description of the Kobuki is in the following section. Attached to the Kobuki are three levels of boards with studs in between to create sturdy balanced platforms. The Kinect 360 was installed on top of the second board and is connected to the Kobuki for the power supply and via USB cable to the netbook. The original netbook which was delivered with the Turtlebot was an Asus X200M and later exchanged for an Acer Travelmate B113 due to malfunction. The Netbook is utilized to store, control, and monitor the software responsible for the proper operation of the SHUSHBot.

Kobuki Unit and Battery

The SHUSHBot needed to be supplied by a variety of power sources. The Kobuki-platform, which is the lowest part of the robot, has a variety of power outlets. These outlets are driven by a rechargeable battery unit in a compartment of the Kobuki illustrated by figure 7.

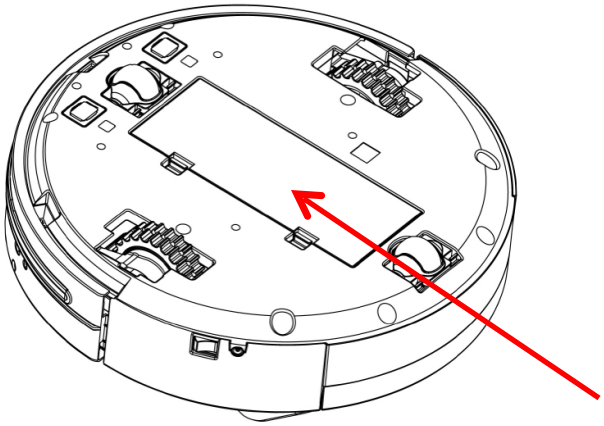


Figure 7. Kobuk unit from underneath [10]

The incorporated battery unit is a 4400mAh Li-Ion battery (4S2P) supplying 14.8V. This battery has an extended life and is stored in the compartment in addition to the 2200mAh standard battery (4S1P). Due to the internal charging circuitry of the Kobuki, it is impossible to hook up both batteries in parallel to achieve even a longer battery time. The 4400mAh battery is plugged into the Kobuki unit. The positive power supply is connected to the emergency shut off switch, which is described in the later section. The large capacity battery has an expected operating time of 7 hours, and the smaller one of 3 hours.

Kobuki Power Distribution

On the front side of the Kobuki there are various outlets. Figure XX displays the Control panel.

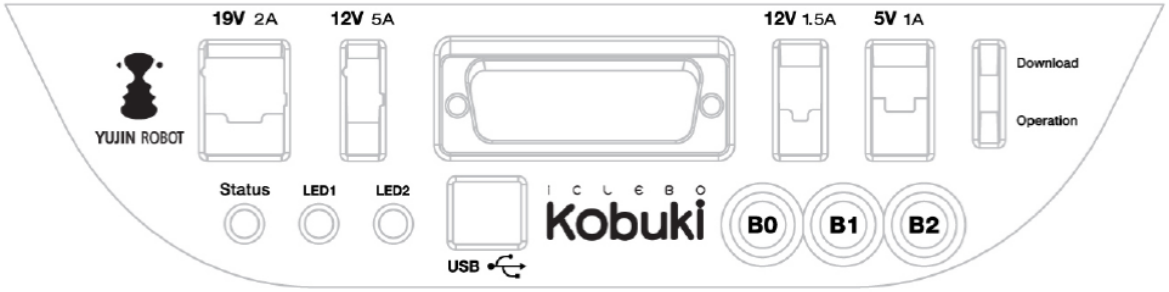


Figure 8. Kobuki Control Panel [10]

There are 3 power connectors: 5V/1A, 12V/1.5A, and 12V/5A. The 12V/1.5A is utilized to power the Kinect. An external wire is connected from the plug to the Kinect. The 12/5A outlet connects via a wire to the speaker amplifier, which is described in section XX. The amplifier needs as maximum of 2W and will not reach the maximum available current. The 5V/1A is used to power the four microphones with their amplifiers, which are illustrated in paragraph XX.

The serial port with its expansion pins: 3.3V/1A, 5V/1A, 4 x analog in, 4 x digital in, 4 x digital out; the programmable two colored LED1 and LED2; as well as the programmable touch buttons B0, B1, and B2 are not used. An integrated audio signal in the Kobuki can be programmed to various beep sequences, which are not modified from its preset conditions. Presently, a three beep audio is played when the Kobuki on and off switch, located at the side of the unit is turned on and when the notebook establishes a connection. This data connection is established with input USB-port via a USB 2.0 cable from the notebook.

The notebook recharging connector provides 19V/2.1A DC. The notebook requires 40W for charging its Lithium Ion battery and for normal operation. In order not to drain the Kobuki battery the connector is disabled when leaving the docking station. The notebook battery lasts for 5 hours and should not run out of power before the SHUSHBot. When the SHUSHBot is in contact with the charge unit it will turn on the recharging connector providing power to the notebook.

Kobuki Docking and Charging Station

In order to maintain a long operation time, the SHUSHBot needs to be recharged periodically. This is achieved by the docking station as shown in figure 9. This docking station will recharge the robot at an expected charging time of 2.6 hours for the large battery and respectively 1.5 hours for the small battery. Additionally, a recharging adapter is integrated with an input of 100-240V, 50/60Hz, 1.5A maximum with a 19VDC, 3.16A output (60W). This adapter is not used.

Figure 9. Kobuk Docking Station [5]

The SHUSHBot will find the docking station with its integrated “Go Home” function. Therefore, it utilizes the Mapping function to find the docking area. This region is defined by a 2x5m area in front of the docking station. Docking IR sensors are programmed to maneuver the robot into the correct position to the recharge connectors. The docking station supplies 19V and a maximum of 3.16A to two connectors. These connectors connect to the Kobuki after it has reached its charge position. The Kobuki maneuvers into the docking space sensing the emitted IR signal. The docking station transmits this IR light by covering three regions: left, central and right, each divided in two sub-fields: near and far. The robot can sense the different frequencies

and determines its own position. The regions and fields are independently identified and can overlap on its borders. In case of the Kobuki being at the middle, it follows the central region's signal and homes into the docking station. Figure 10 displays the different regions.

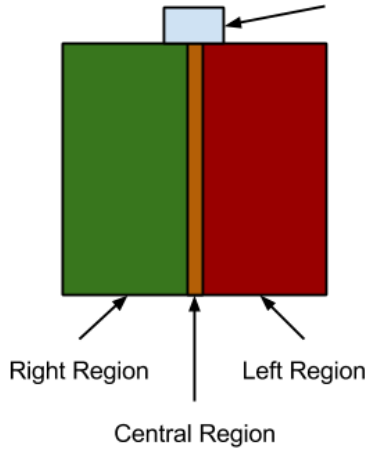


Figure 10. Kobuk Regional Docking [5]

If the robot is to the right, it yaws into the z-direction until the left region signal is detected by the right sensor (data [0]). The robot moves forward until it hits the central region, turning in the negative z-direction to home into the docking station. If the robot is to the right, the opposite procedures apply. Figure 11 shows the used data by the Kobuki platform.

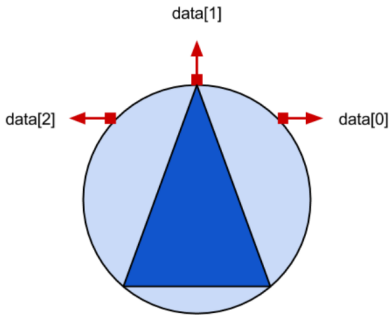


Figure 11. Kobuk Regional Docking [5]

The docking function is integrated in the unit and the Code is not modified for improvement. The Code needs to be loaded and started when launching the software for the SHUSHBot. Figure 12 shows the IR sensors of the Kobuki in green. The sensor data rate is 50Hz and provides enough data to create a smooth transition to the charge unit.

Figure 12. Kobuki with its sensors [10]

Once the SHUSHBot reaches the docking station, two contacts from underneath connect to the charging unit. As long as the contact is not closed, a red light on the docking station is illuminated. Once in contact, the light turns green and the charge logic is started. The logic in the docking station will determine the charge status of the Kobuki battery. The light will flash green as long as the battery is charging and turns steady green once complete. The charging process is independent of the on and off status of the Kobuki unit. The Kobuki LED will flash green as long as the charging process is in progress. This state LED will turn orange to indicate a low battery status. Furthermore, a software program is monitoring the power status of the battery and sending the SHUSHBot to the charging station when needed.

LCD Display:

The Acer netbook is delivered with an 11.6-inch screen in 16:9 widescreen format. The size and resolution of 1366x768 pixels is sufficient for the human interaction function of SHUSHBot. The matte surface distracts reflections and can be used in sunny and shady conditions. However, the brightness with its average luminosity of 180 cd/m^2 is satisfactory only for indoor use. The display appears brighter from upper vertical viewing angles than from the side or lower angles and requires to be tilted at the correct angle when the SHUSHBot is in operation.

External Battery:

The external battery is stored in the battery compartment of the Kobuki and is described in the previous section.

Four Microphone Sound Sensors:

The team ordered four Arduino High Quality Sound sensors designated for detecting high noise levels and helping SHUSHBot maneuver to the area and/or person producing high noise levels. After a long wait the Arduino sensors arrived. They were evaluated for their performance. The sound sensors picked up only sounds within 2 feet distance. Therefore, the team decided to use the microphone itself without the sensor. An amplification circuit needed to be developed.

The function of the microphone sensors is to use the analog output to feed into the analog input of the Arduino. The Arduino algorithm evaluates the voltage peak-to-peak to determine which sensor has the highest noise detection. The front sensor is the main reference. When a side sensor picks up a higher signal, the algorithm publishes a message to ROS. This command moves the SHUSHBot either in the z-yaw direction around its own axis, or it will do a forward turning motion to face the front sensor to the highest signal. This is continued until the SHUSHBot reaches the highest signal strength which is also predefined in the program. Four sensors are needed to detect signal of all four cardinal directions. Every sensor circuit is constructed and designed in the same manner. The amplification circuit is illustrated in figure 13.

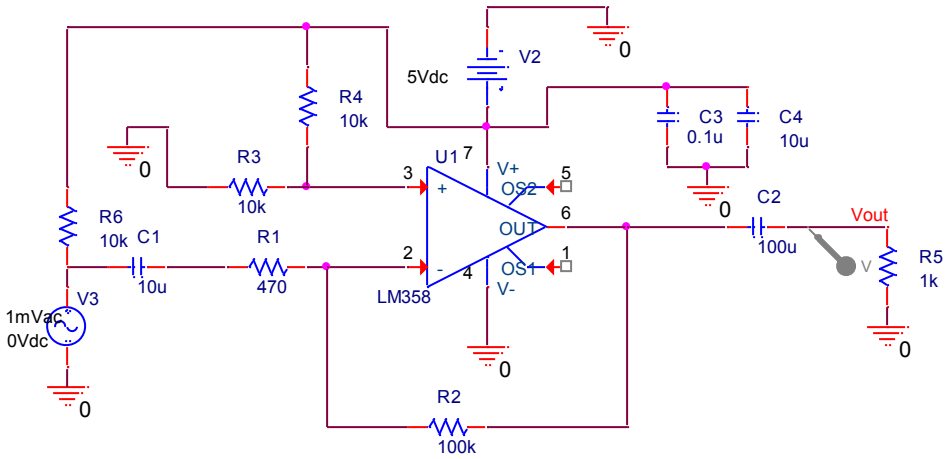


Figure 13. PSPICE Model of Microphone Sensor and Amplifier

The circuit consists of a microphone, an LM358 amplifier, several capacitors, resistors and voltage sources. The following analysis explains the function of the circuit. The microphone is displayed as voltage source V3 for simulation purposes and is at the input of the circuit. The microphone and the amplifier are powered by the 5VDC output of the Kobuki. Initially, it was planned to use the power of the Arduino unit. Although only measured at 1.05mA, the current output of the Arduino could not handle the amplification of the signal. The output of the Kobuki of 1A is definitely able to handle that. A 10µF capacitor is selected to filter high frequencies, a low pass filter, and to decouple the DC voltage. The 470Ω and 100kΩ resistors determine the gain of the inverting amplifier. The gain is G=|-(100k/470K)|=|-213|=46.6dB, which is required to amplify signals/voices up to a distance of about 25 feet. The 100µF capacitor at the output filters low frequencies, acting as a high pass filter. The resulting band pass filter curve can be

seen on figure 14. The normal voice spectrum is between 300 Hz and 3 kHz and is sufficiently covered by the band pass filter.

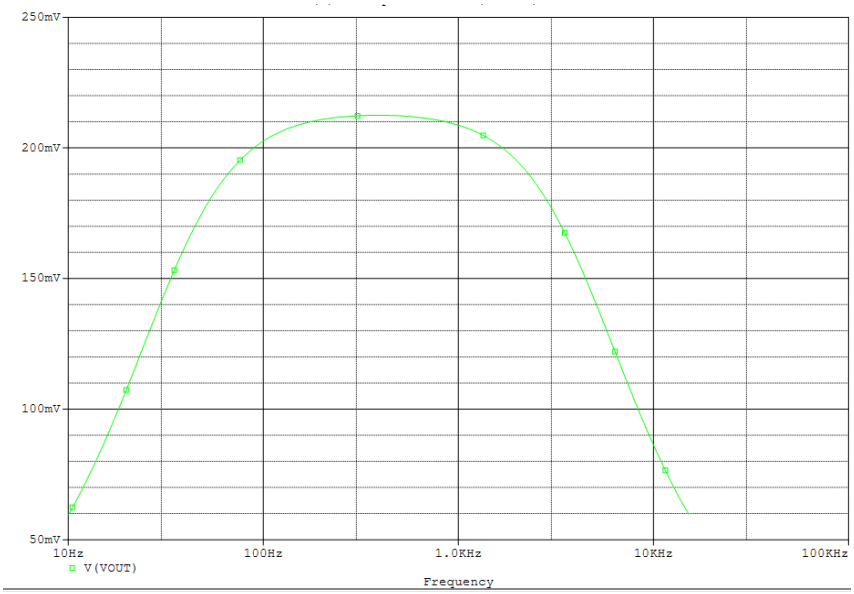


Figure 14. PSPICE Bode Plot of the Microphone circuit

The 1kΩ resistor is used for simulation purposes as the load, which is our analog input into the Arduino. Two 10kΩ resistors at the positive input of the amplifier are used as a voltage divider from the 5VDC input to establish a 2.5VDC to the amplifier. Since the LM358 amplifier is not a real single rail amplifier, the offset is required for the input signal of the microphone. The LM358 was chosen due to the dual amplification possibility, which was eventually not being used. The AC signal is added to the 2.5 VDC in order not to be clipped. The 100µF capacitor decouples this voltage at the output. The rails of the amplifier are fed with the 5VDC and have the common ground of the Kobuki. The positive input rail is supported by two capacitors with 0.1 µF and 10 µF to eliminate any parasitic noise.

The development of the amplifier created a challenge for the team, since the ordered Arduino High Quality Sound sensors did not work as promised. The project was on hold until the sensors were built and incorporated. Most of the algorithm had been completed and needed functional testing. A breadboard solution failed due to vibrations of the SHUSHBot while moving. The team decided to work fast and to solder the microphones on prepared PCBs. A further development of perfect PCBs, which was planned together with the speaker amplifier, would have delayed the project. Therefore, the amplifier for the speaker was created and soldered in a similar fashion.

Amplifier for Speaker

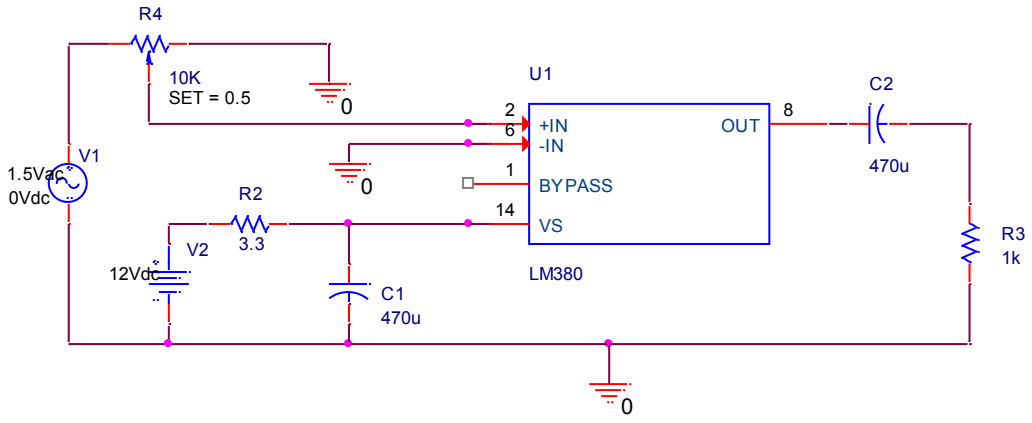


Figure 15. PSPICE Model of LM380 Amplifier

The SHUSHBot needed a speaker in order to facilitate human interaction. The notebook has a build in speaker, which would not be loud enough to silence a speaking person. Therefore, the team decided to add an exterior speaker to the overall design. This speaker is connected to an amplifier based on the LM380. A similar circuit was found on the source http:// cdselectronics.com/kits/Mono Amp.htm. However, the circuit was modified to fit our needs. Figure 15 presents the PSPICE version of the circuit.

The input of the amp utilizes the Speaker outlet of the notebook, which gives a varying AC signal. This signal can be modified in amplitude by a 10kΩ potentiometer. In our circuit we chose a potentiometer which can only be changed with a screw driver to avoid abuse. The potentiometer regulates the gain. The LM380 amplifier has a maximum output of 2W to a connecting speaker. This amplification is achieved by the 12VDC source. The source is connected to the 12V source of the Kabuki platform of the SHUSHBot. The Kabuki platform can handle 5A; and therefore it provides enough power to handle the amplification. The amplifier LM380 is chosen since it achieves a high gain, a low input sensitivity and low distortion. The maximum output of 2W limits the gain, but is sufficient for our circuit. The potentiometer is adjusted that our 1W speaker (chosen for economic reasons) will not reach its limits. The LM 380 amplifier chip is mounted directly onto the PCB, since it can act as a heat sink. The 12V DC is regulated by a protective 3.3Ω resistor. The capacitor at the output is chosen at 470µF to pass the lower frequency range. On the other hand, the 470µF capacitor at the DC input is utilized to decrease high frequencies or noise.

The speaker housing was handcrafted to achieve SHUSHBot's uniqueness. The 2W speaker is tilted towards a person for two reasons. First, it tries to get attention with an order for a speaking person to be silent; and second for communication with attracted students, future students, or other people in the library. The complete speaker is mounted on the second level of the SHUSHBot and creates enough volume.

The speaker amplification exceeded the team's expectations. The clear and loud volume will not be overheard by anyone in the library. It will create the needed attention and facilitate the function of the SHUSHBot.

SHUSHBot Human Interaction Extensions

The Turtlebot2 height measurement is 420 mm. This height is relatively small to create a platform for human interaction. The notebook screen is utilized to display pictures according the need for silencing loud humans or for communicating with interested friendly students. Additionally, the microphone of the notebook needs to sense the voice input of the communicating partner. Therefore, an eight inch extension and an increased top plate are added to the Turtlebot2 base transforming it into a "SHUSHBot." The new height of 630mm allows a better human interaction without sacrificing the center of gravity. The new measurement for the top plate is depicted in figure 16. Several performed tests resulted in no negative performance results. Even the added weights of all added components do not slow down the SHUSHBot. They reduce the Kobuki battery life only by a few minutes.

The top plate added an additional advantage to the design of the SHUSHBot. According to Professor Pranav A. Bhounsule our group was not allowed to do any hardware modifications in the Turtlebot2 due to warranty reasons. The top plate allowed us, to mount all required hardware without changing the Turtlebot2 base. All hardware changes are removable without any noticeable damage to the original base.

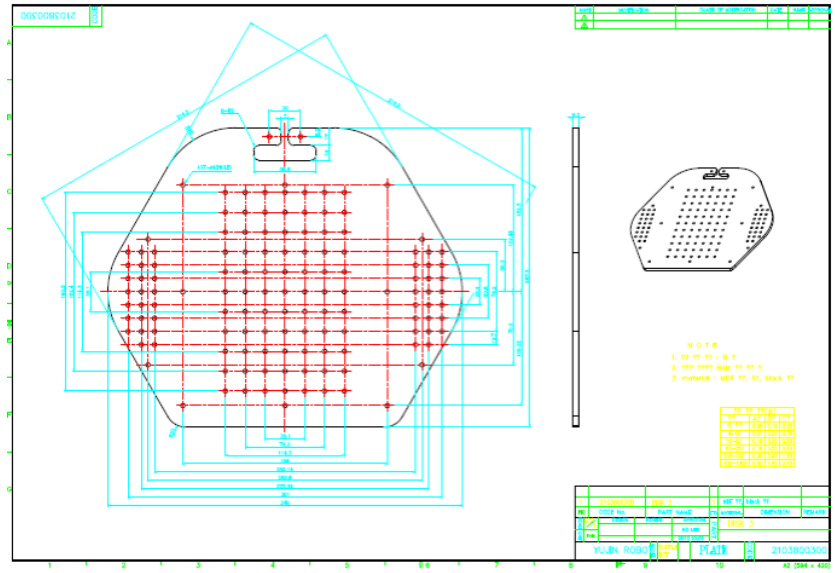


Figure 16. Top extension plate [11]

Attached to the top extension plate are the component box of the Arduino, the speaker amplifier, four microphone sensors, wiring, and protective handles. The emergency shut off switch is positioned and fixed at the rear right to the plate.

Emergency Shut Off Switch

The ANSI/RIA R15.06-1999 American National Standard for Industrial Robots and Robot Systems is obeyed by following the rule 4.6 “every robot shall have stopping functions providing for one or more emergency stop devices.” The robot is not a high powered strong robot. Thus, safety standards are followed for using a newly build robot doing human interaction. An emergency shut off switch is mounted on the top board of the robot next to the notebook. It is highly visible and an easy way for anyone to turn off the robot in case of an emergency, malfunction, or abnormal behavior. The shut off switch interrupts the power supply of the battery to the Kobuki unit by interrupting the anode power cord. However, the notebook power stays on, since we do not see any human harm possible by a computer. This will allow a quick reboot of the system by qualified personal after resetting the power supply. Figure 17 displays the Emergency shut off switch.

Figure 17. Emergency Shutoff Switch

Plexiglas and Protective Handles

Handles are mounted on the top plate to hold the notebook in place to protect it from sliding off when the SHUSHBot has to overcome obstacles like cables. Attached to the handles is a Plexiglas cover to avoid abuse of the notebook keyboard. The Plexiglas is removable by two screws allowing an easy access to the keyboard or to reboot the system.

Software

Software was the major portion of the SHUSHBOT project and was tackled by Team 2. The code for the sound sensors and Arduino was assigned to Thilo and Javier as Javier has the most experience working with the Arduino board. Roberto took on the voice recognition portion of the project because it was based in Java which he has the most experience with. Stacy took the navigation, battery, and listening functions as she has the most experience programming in C ++. Stacy and Javier both worked on the integration portion of the software development.

Start-up Function

As the use of ROS becomes more familiar and more functions are added to complete the assigned task it became obvious the start- up process would need to be simplified. Hence a script has been created to automate the start-up of the robots necessary programs. Files included in the script are the minimal ROS launch file, the file to activate the auto-docking algorithm, the battery monitor program, the Arduino code, the map files, and the main listener program.

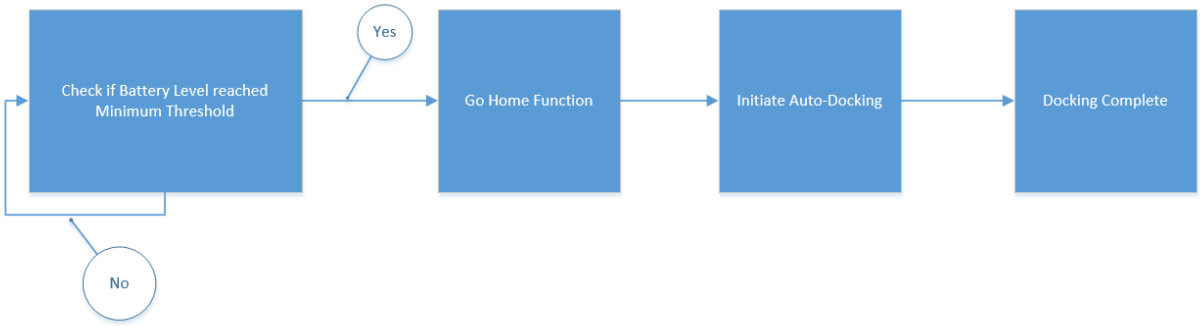


Figure 18. Battery to Docking Flow

Battery Monitor

The battery is monitored by using the Linux ACPI, or advanced configuration and power interface, modules. The battery levels are refreshed at an interval in the state file, the battery monitoring program reads this information in as a string and compares it to what is considered a low battery level. Once the strings match the listener program is signaled to navigate to the coordinates close to the docking station and activate the auto-docking algorithm. The choice to monitor the battery life of the laptop instead of the base was basically the laptop batteries are always depleted first, as the Kobuki base has an extended battery. If not for the constraint of

time, a second section of code would have been added to likewise initiate the auto-dock if the Kobuki base batteries were depleted.

Go Home and Auto Dock Function

This function will be activated after receiving the signal the batteries, either laptop or Kobuki base, are close to being depleted. It will take the pre-determined coordinates from the map which are close to the base and first navigate there. Once the initial destination is reached the Auto docking algorithm is launched commanding the SHUSHBOT to dock and allow for the batteries to fully charge. The reason behind the initial destination coordinates being used exists because the docking algorithm is depended on limited distance infrared sensors. By first having the SHUSHBOT move to the desired coordinates ensures a suitable distance from the charging station is achieved. The Kobuki auto docking package was written by Younghun Ju (yhju@yujinrobot.com) and is licensed as BSD.

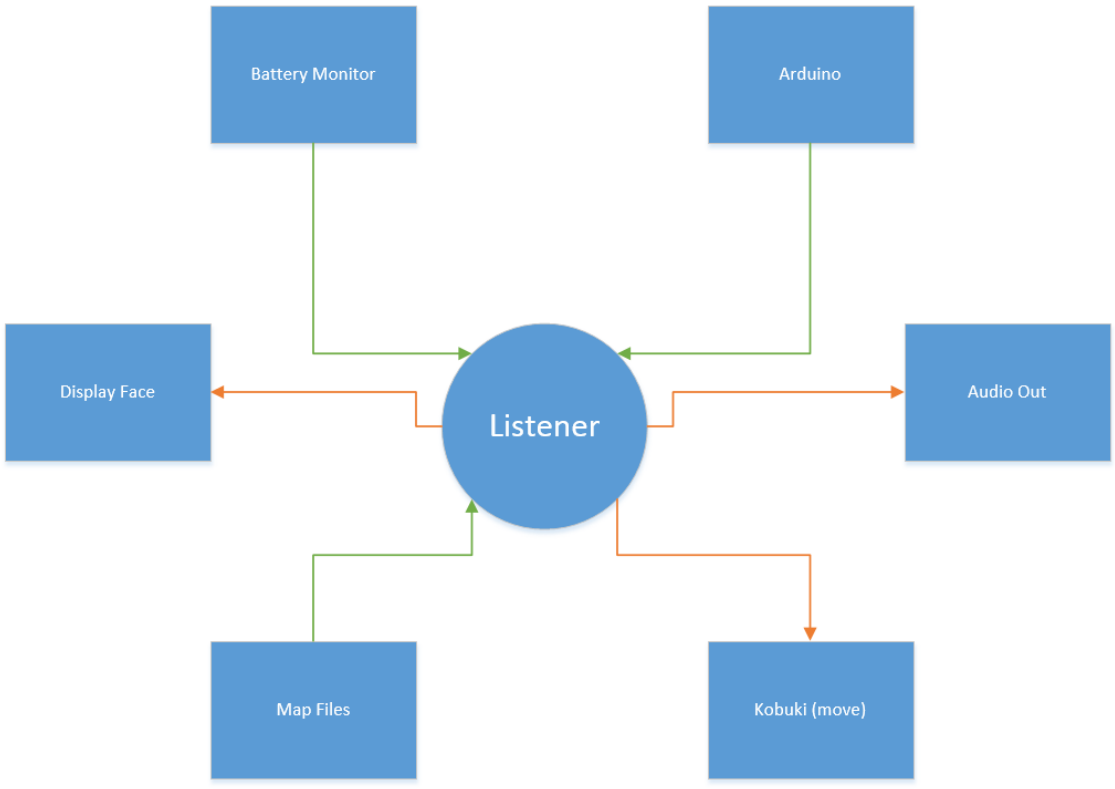


Figure 19. Listener Data Send/Receive

Listener Code

This file is where a majority of our groups programs interact with the robot. In this file he subscribes to the various ROS nodes as dictated by us. When SHUSHBot hears a command it is listening for, it will respond as directed. SHUSHBot will move in the direction requested, if a noise threshold has been reached he will play an audio file reminding the student to be quiet, if the batteries are low SHUSHBot will follow start the go home function, and the location and appropriate times to show the face are in this code.

The listening for direction code will be done using the Publisher/Subscriber method in ROS. When the SHUSHBot receives a command from the sound detection code (which is the Publisher) it has subscribed to it will move in the direction as requested. To direct the base an X-Y coordinate system is used so the commands (Forward, Turn 180, Left, and Right) needed to be converted.

The display functions allow for a face to be displayed on the screen of the TurtleBot laptop, this is to allow for an enhanced experience to improve the human interaction aspect of the project. During normal operation a smiling face will be displayed, when the quiet threshold has been breached, a shushing face, and when the battery is low, a sleeping face. The commands to view these images will be launched from the cpp file that corresponds to the desired expression. The decision to launch them from within the cpp was it being the simplest method for viewing different expressions as a result of the corresponding program.

Mapping and Area 51 Function

Mapping for the SHUSHBot is a multi-step process that was modified from the initial map of the lab reflects the layout of the library. To create the initial map SLAM (Simultaneous localization and mapping) gmapping was used. The first step was to bring up the gmapping software and save the map file to a specified location. Unfortunately, once the map has been created there is not a way to update the map, if updates are needed it would need to be completely recreated. The generated map as saved as two different files, a .yaml file and a .pgm file. The .yaml file contains the location of the map, resolution, origin, negate (whether the white/black free/occupied areas should be reversed), and the occupied and free space estimates.

The .pgm file contains the image of the map as displayed in ROS using the RVIZ program. This map can be edited to exclude areas as we wanted done with our area 51 concept to keep the SHUSHBot from entering the bathroom or getting close to the stair case or any other obstacles where it could be damaged. The main limitation to this is only a certain amount of changes can be made of the SHUSHBot or it will no longer recognize where it is located. Using the Kinect it does a scan of its environment to locate where on the map it is using a Monte Carlo

method, if the SHUSHBot scans and cannot find anything close enough to what it “sees” on the map file it will just keep spinning around and sending errors to ROS that it is stuck. In the images below you can see where the SHUSHBot is on the map, where area 51 is, and a photograph of the actual layout of the mapped area.

Figure 20. Lab (Base Side)

Figure 21. SLAM Map of Lab (Base Side)

Figure 22. Lab (Area 51 Side)

Figure 23. SLAM Map of Lab (Area 51 side)

Arduino/ROS Microphone Sensor Program

The scheme behind this program is to allow the four sensors on SHUSHBot to determine which has a higher incoming noise level and be able to send a command to the Listener to move towards the noise. The Arduino commands must work hand in hand with Stacy's Listener program.

After determining where to position the sensors, we had to make sure that all sensors read the same value when hearing just one sound. Allowing us to makes sure that the computer will not always attend to one direction. The program also includes a thresh max and min, which help the filter random noise from the sensors. For example if our minimum is four volts and maximum is six volts, anything less than four volts coming from the sensors will not move SHUSHBot. Yet anything greater than six will send a command to the listener program to output the silence audio file, meaning the speaker will output a sentence telling the person to be quiet. The block diagram below (figure 24) will help illustrate the sequence in which the Arduino takes each phase carefully and calculates whether or not the incoming sound is important to it or not.

Going through the program in detail is as follows. The incoming sound will be measured by the sensors. From there the program will determine if the signal is strong enough to pursue or to ignore and scan again. If the signal is stronger than the minimum threshold, it will compare to the other sound sensors to determine which direction it should move towards. From there it will filter those signals not needed and take the strong signal to use to advertise to the listener program as well as compare the signal to the max threshold. If breaking the max threshold the program will output a silencing audio file telling the person to be quiet, yet if it is not greater than the threshold then it will simply just repeat the scanning process.

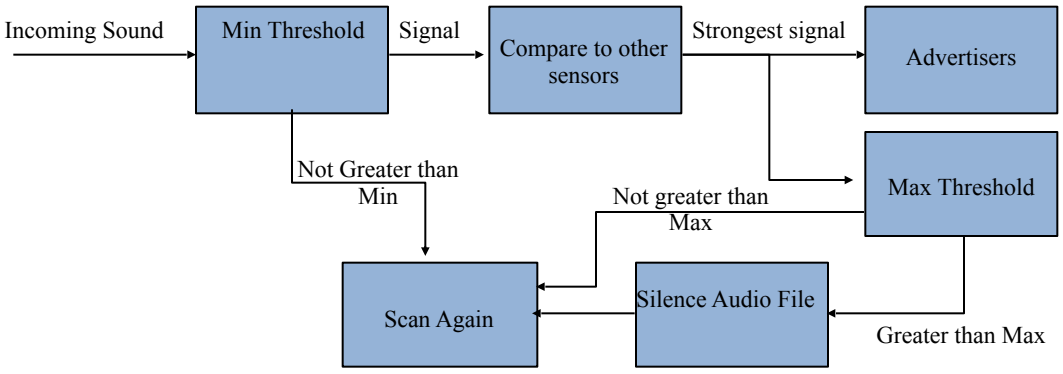


Figure 24. Block diagram of Arduino Program

Human Interaction Program

To have an interactive program we need to be able take in inputs and tie them to an output. To interact with a person we need to have a Graphical User Interface (GUI) that is responsive to the action of the user and can give visual cue. To manage the visuals, program state, and actions we implemented a model-view-controller (MVC) architecture. MVC has three parts that have specific task and are connected as seen in Fig XX. This is the framework on which we build our application for the Human Interactive Program for SHUSHBot (HIPS).

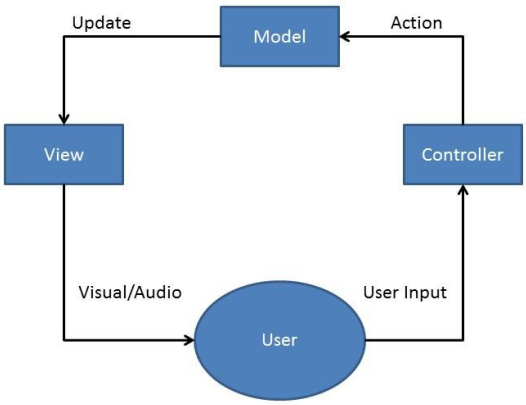


Figure 25. Human Interface Block Diagram

HIPS original purpose was to respond to a person speaking. To give a simple response to what a speaker may have said. For example a speaker may say “Hi” and SHUSHBot would respond with “Hello”. Now our HIPS can tell you the time, recognized names and remember things about you. SHUSHBot uses the Sphinx4 toolkit that is open source from Carnegie Mellon University. Sphinx4 provides several functions that process speech and output a word or a string of words that were spoken. The string output is the result of the best fit match compared to the words in the toolkits dictionary. The string result then goes though the 3 parts of the HIPS program, the controller, model, and view.

The model holds many functions and the current state of the HIPS. The model has states such as if playing audio is true then it will play a corresponding audio file to a corresponding word or string of words that were uttered. It also holds the file names to respective audio, and image files. As well as various functions that help determine what string of words was said and give a corresponding responses. The model calls one function that retrieves the information of a certain user. The information about the user includes their name, likes, and dislikes. This information can be change by the user and/or can also be retrieved by user.

The view is the face of the program. It opens up a simple window which has two parts, an area to display text and an area to display images. This display can be seen in figure 26.

Figure 26. Text image

The view uses the swing class that is slandered in java to open images, display text, and update both images and text.

The controller connects the model and the view. It takes in the string and passes it along to the model. Then the controller calls the view to call the model so that it may do the necessary action need to be done such as displaying new text, or changing the image currently displayed.

Pseudo Code

```
In Main
SetUpMircophone();
OpenGui();
While(GuiIsOpen){
        Result=GetSpeachFromSensor();
```

```
        Controller.heardString(Result);
        View.UpDateView;
}

In Controller
heardString(Result){
Model.ProcessString(Result);
View.Updateinfo(Model.GetResponse())
}

In Model
ProcessString(Result){
Response=MatchResultToResponse();
}

In View
UpdateInfo(Response){
UpdateText(Response);
GetImageFiles();
}
```

An example of how the program works is if the user said “Good Moring”. This string would be sent to the controller which would call the model to process this string. It would then go through several cases to find the appropriate response. All possible strings of words that can be recognizes by the program are hard coded. All possible responses to all possible recognized strings are also hard coded. In our example of “Good Morning” the key word “Good” is used to reduce the number of possible cases. In the current version of the program there are four possible strings that start with “Good”. Once the recognize string is found an appropriate response will be set. If a string response also comes with a correlating image and /or audio file, these files will also be set. All audio files are played inside the model.

Once the model has completed its process it will display any correlating images, play audio files, and give a proper response depending of what was said. When a person introduce themselves by their first name the program will recognized the name and will determine whether this is a new person or a person already in the list of recognized people. If it is a new user the program will begin asking question to the user until the user has answered all the question or the user says “no more questions”. Information about a certain person can also be retrieved by saying “Do you know SOME_PERSON_NAME”. If they are on the list and the user asks

question about that person such as if their favorite color is red and it happens to be true the Shush bot will respond with “Yes they do like the color red”. Here is a block diagram of the program in figure 27.

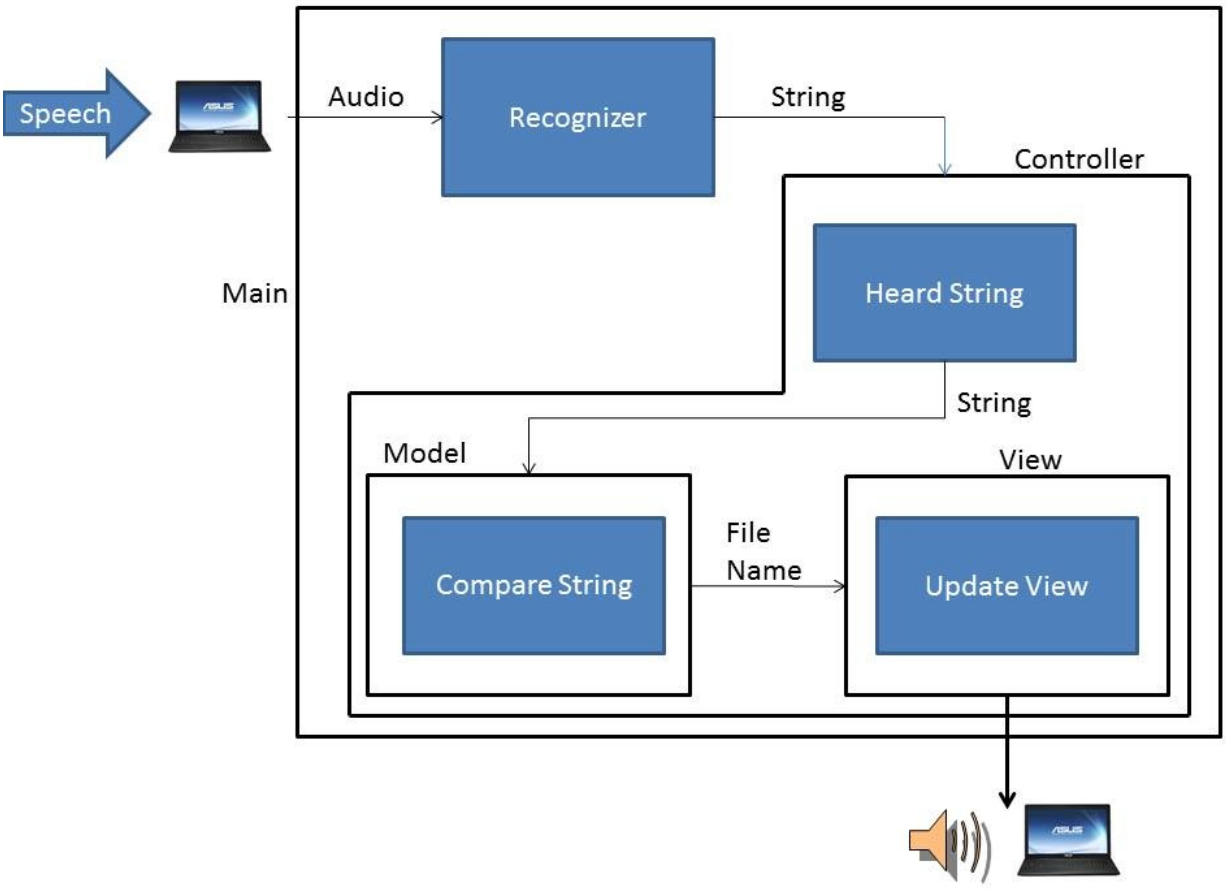


Figure 27. Program Block Diagram

The first problem encountered when implementing the program was in which langue. Original program was meant to be written in C++ but this proved to be difficult. It was difficult to integrate the Sphinx toolkit to work in C++ due to limitation of our programing experience. Also it would have proven difficult to create at GUI since C++ does not have library for graphics. Our solution to these issues was to use Java. Java already has standard libraries for graphics and was easier to plug in the Sphinx4 toolkit since one of our group members has experience in java.

The next issue was being able to display an image without having to resize the window and creating a new window. Through trial and error and following online examples we were able to change images without having to open a new window. Next was plugin in the Sphinx4 tool kit. Using the hello world example took some trials. This was due to lack of knowledge about

having the prober libraries and JAR files need to support the Sphinx4 toolkit. After installing these libraries and referencing them into the program, it began to work properly.

Playing an audio file became an issue. Looking at an example found online it seemed strait forward but proved to be difficult. The issue was that the program was not properly set to the correct audio output. Simply changing one value, the audio was able to play.

Finally after much debugging and trial and error the HIPS program started to come to life. When moving from the Desktop station that it was developed on onto the travelmate laptop there was one major issue. The microphone on the laptop was not performing to our expectations. The onboard microphone was slow and very irresponsible to the user. The program would still run but only after a major delay after someone had spoken. Plugging in an additional microphone resulted in the program being more responsive. With limited time we will sure our separate microphone in our system.

13.0 Major Problems

The biggest overall issue was getting a basic understanding of ROS and the Turtlebot hardware, which lead to several different major obstacles. We were unable to connect the Turtlebot to the Air Rowdy network because of security constraints; to work around this we brought in our own router and created a network for just the Turtlebot and any laptop we were using to communicate with the Turtlebot.

At the request of our sponsoring professor we had the IS department back up the Turtlebot laptop but with their limited experience with Ubuntu they managed to destroy the operating system on the laptop. This resulted in us reinstalling ROS, Ubuntu, and programs we had written, and recreating all of the network settings and options.

At this point the power supply on the laptop failed and a new laptop for the Turtlebot had to be ordered under warranty. Once the new laptop was received we had to reinstall our software and recreate all of the network settings and options again. This laptop was a different model and the battery level could not be read by ROS. ROS expected to battery level to be in a folder called BAT0 but instead the replacement had a folder called BAT1. After contacting the manufacturer, who said because it was software they could not assist us, it was discovered we could modify our. bashrc file to reflect the change in battery folder name and that would allow ROS, which was also started in the terminal, to recognize the battery and monitor the charge.

Along with those laptop problems we had issues with the microphone sensors software. For these sensors we are using Arduino/ROS software to control the output signals that will allow us to determine the direction we must travel to, thus allowing us to tell such person to

silence them. To understand what was needed the team had to get an output from the microphone and be able to see if the sound can be recognized from

14.0 Integration and Implementation

To achieve the goal of both quieting students in the library and human interaction the SHUSHBOT has been designed to run in one of two modes. The first mode is human interactive mode; this is the mode where the SHUSHBOT will communicate with the user with voice recognition. This goal was accomplished by writing two different start-up scripts allowing the mode selection when bringing the SHUSHBOT online.

The second mode is the "quieting" mode; this is the mode where the SHUSHBOT will silence students in the library quiet zone. The main method of communication between the different programs is done using the ROS publisher/subscriber method. The listener program is a subscriber to both the Arduino code and battery monitoring code, which have both been programmed as publishers. The listener repeatedly scans for the following topics:

- Move Right
- Move Left
- Move Forward
- Turn 180 degrees
- Silence
- Low Battery

As soon as one of the topics are heard and recognized, the listener code will go into the subroutine designed for that particular command. If the Battery is Low topic is published the listener program will go into a subroutine that calls the subroutine containing the coordinates close to the location of the charging station and then directs the base to navigate there. Likewise, if any of the direction topics are published by the Arduino code the subroutine will be called to move the base in that precise direction. If the Quiet topic is published by the Arduino code the SHUSHBOT will open a play an audio file politely asking for silence.
The process to integrate the two software components included taking one sensor at a time and testing it individually with the SHUSHBOT to make sure both communication and the correct command was being communicated. Once verified each individual section was functioning correctly with the software, they were all connected and tested as a single sound-sensing unit. The integration process for the auto-docking function was to initial test going to the coordinate and initiating the auto-docking algorithm as a single function. Once verified those two parts functioned together it was time to integrate it with the rest of the listener code. To have the SHUSHBOT stay docked while the batteries fully charged a minimum time limit was needed before the listener program would start accepting commands, which was accomplished by using a sleep command in ROS.

15.0 Comments and Conclusion

Overall the design project has allowed us to expand our knowledge in ROS and robotics itself as well as use our common knowledge to help design SHUSHBot. Thankfully Dr. Pranav was able to fund the project allowing us to really focus on the human interaction and the silencing rather than having pressure building a mobile robot from scratch. As for the team, we were luck to have a brain in each concentration of the electrical engineering department as this allowed us to learn from each other as well as help each other get through what could have been many difficult obstacles.

To conclude our project, there is still much than can be done for future students. The sensors can be higher quality such that they can detect any sound and do not need software amplification and hardware amplification. The Human interaction can always be expanded as well; no robot is ever too human. As mentioned before the purpose of SHUSHBot was not only to silence those in the library but to also encourage students to come to the university and allow them to experience researching robotics along with human interaction.

16.0 Team Members

Javier Gonzalez- team lead as well as programmer for the majority of the Arduino sound sensing program. Javier was in charge of making sure the team was on time with the responsibilities, provided what was needed to complete SHUSHBot, and made sure that the code was developed and integrated to expected final product.

Thilo Janssen- The hardware man, in charge of designing all outer components and making sure that all power distribution was to safety standards and calculated whether or not everything we wanted to attach was possible to do. He built the top layer in which the laptop sits, added all the attachments such as speaker and amplifier, built the sensors with amplifiers attached and established a well-coordinated power distribution to each component.

Roberto Mesquitic- this man is our special programmer, although he has a concentration in signals and systems, he also has a minor in programming. He was given the task to develop the software in which would allow SHUSHBot to be the most “Human” it can be. He fully developed a human interaction program in which the user is able to communicate to the robot and SHUSHBot will respond at the best of its abilities. Along with that he was able to develop a file in which help Stacy output the silence audio file.

Stacy Alexander- Last but not least the “integrator.” Stacy is responsible for most of our programs working all together. She was assigned to develop the program in ROS, something we were all not familiar with, and make SHUSHBot move, map, and collision avoidance, receive incoming silence commands, and outputting the designated audio commands for the interface.

Although she mainly worked with Javier to get the integration moving, most of it was slowly getting together though her programming.

Overall this team is a BEAST! And wouldn't choose any other team.

17.0 References

[1] IEEE Standard (2008). Rechargeable Batteries for Multi-Cell Mobile Computing Devices. pp. c1 - 79.

[2] IEEE Standard 802.11. Information technology -Telecommunications and information exchange between systems Local and metropolitan area networks (2012, Nov 21). pp. 1-2798.

[3] IEEE Standard1680.1-2009. Standard for Environmental Assessment of Personal Computer Products (2010, 3 5). pp. c1-33.

[4] IEEE.org. History of IEEE (2015).

[5] ROS.org. Documentation. 2014-12-19

[6] (2015). sphinx4 (1.0 beta6). Cmusphinx.sourceforge.net.

[7] Various Audio Files [Online]. Soundjax.com.

[8] Various Image Files [Online]. Google.com.

[9] Model-view-controller [Online]. Wikipedia.org.

[10] Kobuki Users Guide

[11] Retrieved from: http://yujinrobot.com/eng/]

Human Interaction Program

Prsmodel

```
package demo1;

public class prsmodel {


	private String NAME;
	//private String SPORT;
	//private String HOBBI;
	private String COLOR,PLACE;
	public int numberOfQuestions=1;//this part you can change Be care full program may crash if you do
	public int onQuestion;
	public boolean answerAllQuestions;

	/**
	 *Question 0 Favorite Color
	 * @param name
	 */
	public prsmodel(String name){
		NAME=name;
		COLOR=null;
		PLACE=null;
		onQuestion=0;
		answerAllQuestions=false;
	}

	public String getName(){
		return NAME;
	}

	public String getCOLOR(){
		return COLOR;
	}
	public String getPlace9(){
		return PLACE;
	}
	public void setPlace(String in){
		PLACE=in;
	}
	public void setCOLOR(String in){
		COLOR=in;
	}

	//public String

}
```

Spconfig

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
  Sphinx-4 Configuration file
-->

<!-- ******************************************************** -->
```

```
<!--  an4 configuration file                    -->
<!-- ******************************************************* -->

<config>

   <!-- ******************************************************* -->
   <!-- frequently tuned properties                          -->
   <!-- ******************************************************* -->

   <property name="logLevel" value="WARNING"/>

   <property name="absoluteBeamWidth"  value="-1"/>
   <property name="relativeBeamWidth"  value="1E-80"/>
   <property name="wordInsertionProbability" value="1E-36"/>
   <property name="languageWeight"     value="8"/>

   <property name="frontend" value="epFrontEnd"/>
   <property name="recognizer" value="recognizer"/>
   <property name="showCreations" value="false"/>


   <!-- ******************************************************* -->
   <!-- word recognizer configuration                        -->
   <!-- ******************************************************* -->

   <component name="recognizer" type="edu.cmu.sphinx.recognizer.Recognizer">
      <property name="decoder" value="decoder"/>
      <propertylist name="monitors">
         <item>accuracyTracker </item>
         <item>speedTracker </item>
         <item>memoryTracker </item>
      </propertylist>
   </component>

   <!-- ******************************************************* -->
   <!-- The Decoder   configuration                          -->
   <!-- ******************************************************* -->

   <component name="decoder" type="edu.cmu.sphinx.decoder.Decoder">
      <property name="searchManager" value="searchManager"/>
   </component>

   <component name="searchManager"
      type="edu.cmu.sphinx.decoder.search.SimpleBreadthFirstSearchManager">
      <property name="logMath" value="logMath"/>
      <property name="linguist" value="flatLinguist"/>
      <property name="pruner" value="trivialPruner"/>
      <property name="scorer" value="threadedScorer"/>
      <property name="activeListFactory" value="activeList"/>
   </component>


   <component name="activeList"
         type="edu.cmu.sphinx.decoder.search.PartitionActiveListFactory">
      <property name="logMath" value="logMath"/>
      <property name="absoluteBeamWidth" value="${absoluteBeamWidth}"/>
      <property name="relativeBeamWidth" value="${relativeBeamWidth}"/>
   </component>
```

```
<component name="trivialPruner"
          type="edu.cmu.sphinx.decoder.pruner.SimplePruner"/>

<component name="threadedScorer"
          type="edu.cmu.sphinx.decoder.scorer.ThreadedAcousticScorer">
   <property name="frontend" value="${frontend}"/>
</component>

<!-- ******************************************************* -->
<!-- The linguist  configuration                             -->
<!-- ******************************************************* -->

<component name="flatLinguist"
          type="edu.cmu.sphinx.linguist.flat.FlatLinguist">
   <property name="logMath" value="logMath"/>
   <property name="grammar" value="jsgfGrammar"/>
   <property name="acousticModel" value="wsj"/>
   <property name="wordInsertionProbability"
          value="${wordInsertionProbability}"/>
   <property name="languageWeight" value="${languageWeight}"/>
   <property name="unitManager" value="unitManager"/>
</component>

<!-- ******************************************************* -->
<!-- The Grammar  configuration                              -->
<!-- ******************************************************* -->

<component name="jsgfGrammar" type="edu.cmu.sphinx.jsgf.JSGFGrammar">
   <property name="dictionary" value="dictionary"/>
   <property name="grammarLocation"
       value="resource:/demo1/"/>
   <property name="grammarName" value="sp"/>
        <property name="logMath" value="logMath"/>
</component>


<!-- ******************************************************* -->
<!-- The Dictionary configuration                            -->
<!-- ******************************************************* -->

<component name="dictionary"
   type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
   <property name="dictionaryPath"
          value="resource:/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/cmudict.0.6d"/>
   <property name="fillerPath"
          value="resource:/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/noisedict"/>
   <property name="addSilEndingPronunciation" value="false"/>
   <property name="allowMissingWords" value="false"/>
   <property name="unitManager" value="unitManager"/>
</component>

<!-- ******************************************************* -->
<!-- The acoustic model configuration                        -->
<!-- ******************************************************* -->
<component name="wsj"
          type="edu.cmu.sphinx.linguist.acoustic.tiedstate.TiedStateAcousticModel">
   <property name="loader" value="wsjLoader"/>
```

```
    <property name="unitManager" value="unitManager"/>
</component>

<component name="wsjLoader" type="edu.cmu.sphinx.linguist.acoustic.tiedstate.Sphinx3Loader">
    <property name="logMath" value="logMath"/>
    <property name="unitManager" value="unitManager"/>
    <property name="location" value="resource:/WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz"/>
</component>


<!-- ******************************************************** -->
<!-- The unit manager configuration                           -->
<!-- ******************************************************** -->

<component name="unitManager"
    type="edu.cmu.sphinx.linguist.acoustic.UnitManager"/>

<!-- ******************************************************** -->
<!-- The frontend configuration                               -->
<!-- ******************************************************** -->

<component name="frontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
    <propertylist name="pipeline">
        <item>microphone </item>
        <item>preemphasizer </item>
        <item>windower </item>
        <item>fft </item>
        <item>melFilterBank </item>
        <item>dct </item>
        <item>liveCMN </item>
        <item>featureExtraction </item>
    </propertylist>
</component>

<!-- ******************************************************** -->
<!-- The live frontend configuration                          -->
<!-- ******************************************************** -->
<component name="epFrontEnd" type="edu.cmu.sphinx.frontend.FrontEnd">
    <propertylist name="pipeline">
        <item>microphone </item>
        <item>dataBlocker </item>
        <item>speechClassifier </item>
        <item>speechMarker </item>
        <item>nonSpeechDataFilter </item>
        <item>preemphasizer </item>
        <item>windower </item>
        <item>fft </item>
        <item>melFilterBank </item>
        <item>dct </item>
        <item>liveCMN </item>
        <item>featureExtraction </item>
    </propertylist>
</component>

<!-- ******************************************************** -->
<!-- The frontend pipelines                                   -->
<!-- ******************************************************** -->
```

```
<component name="dataBlocker" type="edu.cmu.sphinx.frontend.DataBlocker">
   <!--<property name="blockSizeMs" value="10"/>-->
</component>

<component name="speechClassifier"
         type="edu.cmu.sphinx.frontend.endpoint.SpeechClassifier">
   <property name="threshold" value="13"/>
</component>

<component name="nonSpeechDataFilter"
         type="edu.cmu.sphinx.frontend.endpoint.NonSpeechDataFilter"/>

<component name="speechMarker"
         type="edu.cmu.sphinx.frontend.endpoint.SpeechMarker" >
   <property name="speechTrailer" value="50"/>
</component>


<component name="preemphasizer"
         type="edu.cmu.sphinx.frontend.filter.Preemphasizer"/>

<component name="windower"
         type="edu.cmu.sphinx.frontend.window.RaisedCosineWindower">
</component>

<component name="fft"
       type="edu.cmu.sphinx.frontend.transform.DiscreteFourierTransform">
</component>

<component name="melFilterBank"
   type="edu.cmu.sphinx.frontend.frequencywarp.MelFrequencyFilterBank">
</component>

<component name="dct"
       type="edu.cmu.sphinx.frontend.transform.DiscreteCosineTransform"/>

<component name="liveCMN"
         type="edu.cmu.sphinx.frontend.feature.LiveCMN"/>

<component name="featureExtraction"
         type="edu.cmu.sphinx.frontend.feature.DeltasFeatureExtractor"/>

<component name="microphone"
         type="edu.cmu.sphinx.frontend.util.Microphone">
   <property name="closeBetweenUtterances" value="false"/>
</component>


<!-- ******************************************************* -->
<!--  monitors                                               -->
<!-- ******************************************************* -->

<component name="accuracyTracker"
          type="edu.cmu.sphinx.instrumentation.BestPathAccuracyTracker">
   <property name="recognizer" value="${recognizer}"/>
   <property name="showAlignedResults" value="false"/>
   <property name="showRawResults" value="false"/>
</component>
```

```
    <component name="memoryTracker"
               type="edu.cmu.sphinx.instrumentation.MemoryTracker">
        <property name="recognizer" value="${recognizer}"/>
            <property name="showSummary" value="false"/>
            <property name="showDetails" value="false"/>
    </component>

    <component name="speedTracker"
               type="edu.cmu.sphinx.instrumentation.SpeedTracker">
        <property name="recognizer" value="${recognizer}"/>
        <property name="frontend" value="${frontend}"/>
            <property name="showSummary" value="true"/>
            <property name="showDetails" value="false"/>
    </component>


    <!-- ******************************************************* -->
    <!--  Miscellaneous components                               -->
    <!-- ******************************************************* -->

    <component name="logMath" type="edu.cmu.sphinx.util.LogMath">
        <property name="logBase" value="1.0001"/>
        <property name="useAddTable" value="true"/>
    </component>

</config>
```

Spcontroller

```
package demo1;

public class spcontroller {

	private spview view;
	private spmodel model;


	public spcontroller(spview view,spmodel model){
		this.model=model;
		this.view=view;
	}

	/**
	 * This Function takes in a string variable
	 * Passes it to model.setImage and the calls
	 * view.updateImage using the new image set my the model
	 */
	public void heardString(String in){
		System.out.println("In SPCrtl heard: "+in);
		model.setImageName(in);
		model.procesString(in);

		view.updateImage(model.getImageName(),model.getResponse());
		model.playAudio();
		view.mainFrame.repaint();

	}
```

}

Spgram

```
#JSGF V1.0;

/**
 * JSGF Grammar for Hello World example
 */

grammar jsgf;

/*public <greet> = (Good morning | Hello) ( Bhiksha | Evandro | Paul | Philip | Rita | Will );*/

/*public <basicCmd> = <startPolite> <command> <endPolite>;
<command> =<action> <object>;
<action> = /10/ open | /2/ close | /1/ delete | /1/ move;
<object>=[the | a ] (window | file | menu);
<startPolite> = (please | kindly | could you | oh mighty computer)*;
<endPolite> =[please | thanks | thank you];

Proper title <SomethingSomething> and not <Something Something> No spaces!!!!

*/

public <word>=<BasicTalk>;
<BasicTalk>= <Greet> | <Question> | <simpleRep> |<name>| <song> | <aware> | <Farewell>;
<Greet>= <dayoftime> | <hello>;
<dayoftime> = [good] (morning | afternoon | evening );
<hello> = (hello | hello friend | hello buddy | greetings );
<Question>= <time> | <doing> | <me> | <color> ;
<aware>=(are you self aware);
<color>=(red | blue | green | yellow | purple | white | black | pink );


<me>=<myname>| <love>;
<love>=(what is love| what are their names | their names| hello it is pranav | next );
<simpleRep>= (yes | maybe | ok | no);
<myname>=(what is your name | who are you | you are| who made you);
<time> = (what time is it| do you have the time | time | what is the time);
<doing> = (how are you doing | are you doing well | doing well | what is your purpose | who is your savior | now
what);
<Farewell>= (goodbye | bye | good night | night | farewell | later);
<song>=(baby don't hurt me);
<name>=(my name is| name is| name ) (roberto | javier | stacy | thilo);
```

Spmain

```
package demo1;

import javax.swing.JFrame;

import edu.cmu.sphinx.frontend.util.Microphone;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
```

```
public class spmain {
        public static void main(String[] args){
                /**
                 * This code follows from the hello world example with
                 * additions made to it.
                 */

                spmodel model=new spmodel();
                spview view =new spview();
                //Conects the spmodel to the spview
                spcontroller controller =new spcontroller(view,model);

                ConfigurationManager cm;

    if (args.length > 0) {
       cm = new ConfigurationManager(args[0]);
    } else {
       cm = new ConfigurationManager(spmain.class.getResource("sp.config.xml"));
    }

    Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
    recognizer.allocate();


    // start the microphone or exit if the programm if this is not possible
    Microphone microphone = (Microphone) cm.lookup("microphone");
    if (!microphone.startRecording()) {
       System.out.println("Cannot start microphone.");
       recognizer.deallocate();
       System.exit(1);
    }


    view.mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    view.mainFrame.setSize(620, 512);
    view.mainFrame.setVisible(true);

    // loop the recognition until the window is not visible.
    // My Additions
    while (view.mainFrame.isVisible()) {
       System.out.println("Start speaking. Close window to quit.\n");

       Result result = recognizer.recognize();

       if (result != null) {
          String resultText = result.getBestFinalResultNoFiller();
          System.out.println("You Said: "+resultText);
          controller.heardString(resultText);
          view.mainFrame.repaint();
       } else {
          System.out.println("I can't hear what you said.\n");
       }
    }
        }
}
```

Spmodel

```
package demo1;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.GregorianCalendar;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.DataLine;
import javax.sound.sampled.LineUnavailableException;
import javax.sound.sampled.Mixer;
import javax.sound.sampled.UnsupportedAudioFileException;




public class spmodel {

	/*the rodwy head image is the default for the start of the program*/
	private String[] imgFiles={"UTSA.jpg","dog4.jpg","horse.jpg","tiger.jpg","trtl.jpg","bird2.jpg"};
	private String[] imgCommand={"dog","horse","tiger","turtle","bird"};
	private String[]
audioFiles={"hello3.wav","Doing_Well.wav","Farewell_Friend.wav","Good_Morning.wav","Good_Afternoon.wav
","Good_Evening.wav","Good_Night.wav"
                                        ,"Ha_Ha_Ha.wav","I_Am_Doing_Fine.wav","Yes.wav","Goodbye.wav","A
re_You_There.wav","A_Team_Of_Electrical_Engineers.wav","My_Name_Is_Shushbot.wav"
                                        ,"Self_Aware_Reply.wav","Good_That_Makes_Two_Of_Us.wav","Reply_T
o_Breathing_Right_Now.wav","My_Creators_Are_Names.wav","Dr_Pranav.wav"
                                        ,"Baby_Don_39_t_Hurt_Me.wav","Purpose.wav","Ultron.wav","Presention
_Is_Over.wav"};
	private String[] listOfQuestions={"What is your Favorite Color?","Where are you from?","Is it nice
there?",};
	private String response;


	private int image_number;
	private int audio_number;
	private int talkingToPerson_i;
	private boolean playAudio;
	private boolean questionAsked;
	private boolean questions;
	private boolean conversation;
	private boolean aware;
	private boolean made;
	private boolean yesq;

	private ArrayList<prsmodel> listOfPeople;


	public static Mixer mixer;
	public static Clip clip;

	/**
	 * Initial Set up of model
```

```
 */
public spmodel(){
	playAudio=false;
	image_number=0;
	audio_number=0;
	listOfPeople=new ArrayList<prsmodel>();

	questions=false;
	questionAsked=false;
	conversation=false;
	aware=false;
	made=false;
	yesq=false;
	Mixer.Info[] mixInfo = AudioSystem.getMixerInfo();
	for(Mixer.Info info: mixInfo){
		System.out.println(info.getName()+"    "+info.getDescription());
	}


	mixer = AudioSystem.getMixer(mixInfo[1]);

	DataLine.Info dataInfo= new DataLine.Info(Clip.class, null);
	try{
		clip = (Clip) mixer.getLine(dataInfo);
	}catch(LineUnavailableException e){
		e.printStackTrace();
	}


}

/**
 * Return the image name base of the image_number
 * @return
 */
public String getImageName(){
	return imgFiles[image_number];
}


/**
 * this function compares the String in to the imgCommand.
 * If it finds a match it will return the i value + one to the
 * Corresponding imgNameList.
 * If it cannot find anything it will default to 0;
 * @param in
 */
public void setImageName(String in){
	System.out.println("In Model : "+in);
	System.out.println(in.length());
	in.toLowerCase();
	int i;
	//Fix this loop because its wrong!!!! :(
	for(i=0;i<5;i++){

		if(in.equals(imgCommand[i])){
			image_number=i+1;
		}
```

```
		}
		if(i==6){
			image_number=0;
		}

}/*end of setimageName*/

public String getResponse(){
		return response;
}

public void procesString(String in){
		in.toLowerCase();
		if(conversation){
			if(questions){
				if(yesq){
					if(questionAsked){
						answerToQuestion(in);

					}
					else {
						askQuestions();
					}
				}
				else if (in.equals("yes")|in.equals("ok")){
					if(!yesq){
						yesq=true;
						if(questionAsked){
							answerToQuestion(in);

						}
						else {
							askQuestions();
						}
					}

				}
				else if(in.equals("no")){
					conversation=false;
					questions=false;
					yesq=false;
					response="Ok, ask me something else.";
				}
				else {
					response="I did not hear you correctly";
				}
			}
			else if(aware){
				if(in.equals("yes")||in.equals("maybe")){
					conversation=false;
					aware=false;
					playAudio=true;
					audio_number=15;
				}
				else if(in.equals("no")){
					playAudio=true;
					audio_number=16;
					response="You are now breathing manually";
```

```
                        }
                    }
                    else if(made){
                        if(in.equals("what are their names")||in.equals("their names")){
                            playAudio=true;
                            audio_number=17;
                            conversation=false;
                            made=false;
                        }
                        else {
                            conversation=false;
                            made=false;
                            response="Im stuck, say anything.";
                        }

                    }
                }

                else if(ifName(in));

                else greetings(in);
            }



            /**
             *
             * @param Answer
             */
            private void answerToQuestion(String Answer){
                if(Answer.equals("no more questions")){//Stop asking question and return to normal mode
                    questions=false;
                    questionAsked=false;
                    conversation=false;
                    yesq=false;
                    response="Ok no more Questions";
                }
                else if(listOfPeople.get(talkingToPerson_i).answerAllQuestions){
                    response="Questions are Done, say \"Next\"";
                }
                else {
                    if(listOfPeople.get(talkingToPerson_i).onQuestion==0){
                    listOfPeople.get(talkingToPerson_i).setCOLOR(Answer);
                    response="Your Favorite Color is "+listOfPeople.get(talkingToPerson_i).getCOLOR()+".
Say \"next\"";
                    questionAsked=false;
                    listOfPeople.get(talkingToPerson_i).onQuestion++;
                    }



                }

            }


            /**
```

```
	 * This function sets the response to a question in the listOfQuestions array in the model
	 * Based on the number of question available from the prsmodel
	 * it sets questionsAsekd to true so that the next string is the answer to the questioned asked
	 */
	private void askQuestions(){

if(listOfPeople.get(talkingToPerson_i).numberOfQuestions==listOfPeople.get(talkingToPerson_i).onQuestion){
				listOfPeople.get(talkingToPerson_i).answerAllQuestions=true;
				questions=false;
				conversation=false;
				yesq=false;
				response="Question are done, say something else.";
			}
			else {
				response=listOfQuestions[listOfPeople.get(talkingToPerson_i).onQuestion];
				questionAsked=true;
			}
	}


	/**
	 * checkNames checks the list of people in listOfPeople
	 * if the list is empty then it will create a new person and add them.
	 * if the name is in the list it will turn question to true.  When "question"
	 * is true it will ask the user questions until it is done or the user says no more questions
	 *
	 * if the person is found it will create a new person and add them.
	 *
	 * in all case the program will go to the ask questions state whenever questions is true
	 *
	 */
	private void checkNames(String name){
		if(listOfPeople.isEmpty()){
			prsmodel person=new prsmodel(name);
			listOfPeople.add(person);
			response=response+" Can i ask you questions?";
			conversation=true;
			questions=true;//Start asking question about person and will stay in this mode until all
question are answered
		}
		else{
			boolean personFound=false;
			int size=listOfPeople.size();
			for(int i=0;i<size;i++){
				if(listOfPeople.get(i).getName().equals(name)){
					personFound=true;
					talkingToPerson_i=i;
					response="Hello "+listOfPeople.get(i).getName();
					if(!listOfPeople.get(talkingToPerson_i).answerAllQuestions){
						response=response+" Can i ask you questions?";
						conversation=true;
						questions=true;
					}

					break;
				}
			}
```

```
                    if(!personFound){
                            prsmodel person=new prsmodel(name);
                            listOfPeople.add(person);
                            talkingToPerson_i=listOfPeople.size()-1;
                            response=response+" Can i ask you questions?";

                    }
            }
}


/**
 * isName checks if a name was said.
 * So far the name are only a few butcheckNames(temp[len-1]); will grow large
 * If a name is found it will return true otherwise false
 * For expanding the idea maybe call other function that
 * have the names in alphabet order
 * @param in
 * @return
 */
private boolean ifName(String in){
        if(in.contains("name")&&!in.contains("your")){
                String temp[]=in.split("\\s+");
                int len=temp.length;

                if(temp[len-1].startsWith("j")){
                        if(nameStartWith_J(temp[len-1])){
                                checkNames(temp[len-1]);
                                return true;
                        }
                        return false;
                }
                else if(temp[len-1].startsWith("r")){
                        if(nameStartWith_R(temp[len-1])){
                                checkNames(temp[len-1]);
                                return true;
                        }
                        return false;
                }
                else if(temp[len-1].startsWith("s")){
                        if(nameStartWith_S(temp[len-1])){
                                checkNames(temp[len-1]);
                                return true;
                        }
                        return false;
                }
                else if(temp[len-1].startsWith("t")){
                        if(nameStartWith_T(temp[len-1])){
                                checkNames(temp[len-1]);
                                return true;
                        }
                        return false;
                }


        }

        return false;
```

```
}

/**
 * the following nameStartWith_(letter)
 * compares all possible names that start with some letter and return
 * true if a name is found
 * false otherwise
 *
 * @param in
 * @return
 */


private boolean nameStartWith_J(String in){
        if(in.equals("javier")){
                response="Hello Javier ;)";
                return true;
        }
        return false;
}
private boolean nameStartWith_R(String in){
        if(in.equals("roberto")){
                response="Hello Roberto";
                return true;
        }
        return false;
}
private boolean nameStartWith_S(String in){
        if(in.equals("stacy")){
                response="Hello Stacy";
                return true;
        }
        return false;
}
private boolean nameStartWith_T(String in){
        if(in.equals("thilo")){
                response="Hello Thilo";
                return true;
        }
        return false;
}



/**\
 * response will compare a String in to various other strings
 * if their is a match it will set response to that string
 * Other actions can be made such as playAudio or displaying an image.
 * @param in
 */
private void greetings(String in){
        //in.toLowerCase();

        if(in.startsWith("hello")){
                if(in.equals("hello")){
                        playAudio=true;
                        audio_number=0;
                        response="Hello!";
```

```
            }
            else if(in.equals("hello it is pranav")){
                        playAudio=true;
                        audio_number=18;
                        response="Hello Dr. Pranav";
            }

            else{
                        response="I could not hear you.";
            }

}
else if(in.equals("what is your purpose")){
            playAudio=true;
            audio_number=20;
            response="World Domination!";
}
else if(in.equals("who is your savior")){
            playAudio=true;
            audio_number=21;
            response="Ultron!!!";
}
else if(in.equals("greetings")){
            response="Greetings!";
}
else if(in.equals("now what")){
            playAudio=true;
            audio_number=22;
            response="Senior Design is Over!!!";
}
else if(in.startsWith("good")&&!in.equals("goodbye")){
            if(in.equals("good morning")){
                        playAudio=true;
                        audio_number=3;
                        response="Good Morning";
            }
            else if(in.equals("good afternoon")){
                        playAudio=true;
                        audio_number=4;
                        response="Good Afternoon";
            }
            else if(in.equals("good evening")){
                        playAudio=true;
                        audio_number=5;
                        response="Good evening to you.";
            }
            else{
                        playAudio=true;
                        audio_number=6;
                        response="Good night";
            }
}
else if(in.equals("what is your name")){
            playAudio=true;
            audio_number=13;
            response="I am Shushbot";
}
else if(in.contains("who")){
```

```
		if(in.equals("who made you")){
			playAudio=true;
			audio_number=12;
			response="Senior Eletrical Engineering Students";
			conversation=true;
			made=true;
			}
		else if(in.equals("who are you")){
			playAudio=true;
			audio_number=13;
			response="I am Shushbot";

		}
}

else if(in.contains("are")){
		if(in.contains("how")){
			if(in.equals("how are you")){
				response="Im doing well thank you for asking.";
			}
			else if(in.equals("how are you doing")){
				playAudio=true;
				audio_number=1;
				response="Doing well";
			}

		}
		else if(in.equals("are you self aware")){
			conversation=true;
			aware=true;
			playAudio=true;
			audio_number=14;
			response="Hard to say, are you?";
		}
		else if(in.equals("are you doing well")){
			playAudio=true;
			audio_number=8;
			response="Im doing fine, thanks!";
		}
		else {
			response="what did you say?";
		}
}


else if(in.equals("doing well")){
		playAudio=true;
		audio_number=9;
		response="Yes!!!";
}
else if(in.equals("what is love")){
		playAudio=true;
		audio_number=19;
		response="Dont hurt me no more";
}

else if(in.equals("goodbye")){
		playAudio=true;
```

```
                audio_number=2;
                response="Farewell friend";
        }
        else if(in.equals("farewell")||in.equals("bye")){
                playAudio=true;
                audio_number=10;
                response="Goodbye";
        }
        else if(in.equals("later")){
                response="Later homes!!!";
        }

        else if(in.contains("time")){
                response="The time is"+getTime();

        }
        else{
                playAudio=true;
                audio_number=11;
                response="Are you their?";
        }
}


/**
 * Gets the time from the Computer
 * @return a string of the time.
 */
private String getTime(){
        GregorianCalendar time=new GregorianCalendar();
        int hour=time.get(Calendar.HOUR);
        int min=time.get(Calendar.MINUTE);
        int dayoftime=time.get(Calendar.AM_PM);
        String mint=min+"";
        if(min<10){
                mint="0"+mint;
        }
        if(hour==0){//Zero is 12
                hour=12;
        }
        if(dayoftime==1){
                return " "+hour+":"+mint+" PM";
        }
        return " "+hour+":"+mint+" AM";
}


/**
 * Plays audio file if playAudio is true
 */
public void playAudio(){
        if(playAudio){

                //read in file
                try {

                        String getFile="myAudio/"+audioFiles[audio_number];//Keep all files in
myaudio
```

```
                                        File soundFile=new File(getFile);
                                        AudioInputStream audioStream =
AudioSystem.getAudioInputStream(soundFile);
                                        clip.open(audioStream);

                                }catch(LineUnavailableException e){
                                        e.printStackTrace();
                                }catch(UnsupportedAudioFileException u){
                                        u.printStackTrace();
                                }catch(IOException i){
                                        i.printStackTrace();
                                }

                                //Play file
                                clip.start();
                                do{
                                        try{
                                                Thread.sleep(50);
                                        }catch(InterruptedException ie){
                                                ie.printStackTrace();

                                        }
                                }while(clip.isActive());
                                clip.stop();
                                clip.close();//Close file
                                playAudio=false;
                        }
                }
}
```

Spview

```
package demo1;


import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;

import javax.imageio.ImageIO;
import javax.swing.BoxLayout;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

public class spview extends JFrame{

        /**
         *
         */
        private static final long serialVersionUID = 1L;
        public JFrame mainFrame;
        private String myImage="UTSA.jpg";
        private JLabel label;
        private JLabel textLabel;
        private JPanel mainPanel;
```

```
private String textString;
/**
 * Initiales the view
 */
public spview(){


        mainFrame=new JFrame("Human Interaction");

        label=new JLabel();
        mainPanel=new JPanel();
        mainPanel.setLayout(new BoxLayout(mainPanel,BoxLayout.X_AXIS));

        textString="Start Talking :)";

        textLabel = new JLabel(textString);
        drawMyImage();
        mainPanel.add(label);
        mainPanel.add(textLabel);
        mainFrame.add(mainPanel);

}

/**
 * Draws the new image in the same frame
 */
private void drawMyImage(){

        BufferedImage img=null;
        try{
                img=ImageIO.read(new File("myImg/"+myImage));
        }
        catch(Exception e){
                System.out.println("spview: Error did not open \n"+e);
        }
        BufferedImage img2= new BufferedImage(256,256,BufferedImage.TYPE_INT_RGB);
        Graphics2D g=img2.createGraphics();

        g.drawImage(img,0,0,256,256,null);
        g.dispose();

        ImageIcon imgIcon=new ImageIcon(img2);

        label.repaint();
        label.setIcon(imgIcon);


        //mainFrame.getContentPane().add(label,BorderLayout.CENTER);
        //mainPanel.add(label);

}

/**
 * updates the image in the frame
 */
public void updateImage(String in,String text){
        System.out.println("In view heard :"+in+"\n ");
        myImage=in;
```

```
                System.out.println(myImage);
                drawMyImage();
                textLabel.setText(text);
                //mainPanel.add(textLabel);


        }
}
```

Listener/Mobile Program

Listener Code

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>

typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;

void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
 ROS_INFO("I am going to move [%s]", msg->data.c_str());
 MoveBaseClient ac("move_base", true);
 while(!ac.waitForServer(ros::Duration(5.0))){
  ROS_INFO("Check your map file path if More than 5 Seconds!");
 }
 move_base_msgs::MoveBaseGoal goal;
 goal.target_pose.header.frame_id = "base_link";
 goal.target_pose.header.stamp = ros::Time::now();
 goal.target_pose.pose.position.y = .5;
 goal.target_pose.pose.orientation.w = .5;
 ac.sendGoal(goal);
 ac.waitForResult();
 if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
        ROS_INFO("Hooray, I moved in the +y Direction!");
}

void chatterCallback2(const std_msgs::String::ConstPtr& msg)
{
 ROS_INFO("I will now [%s] Degrees ", msg->data.c_str());
 MoveBaseClient ac("move_base", true);
 while(!ac.waitForServer(ros::Duration(5.0))){
  ROS_INFO("Check your map file path if More than 5 Seconds!");
 }
 move_base_msgs::MoveBaseGoal goal;
 goal.target_pose.header.frame_id = "base_link";
 goal.target_pose.header.stamp = ros::Time::now();
 goal.target_pose.pose.position.y = -.5;
 goal.target_pose.pose.orientation.w = ;
 ac.sendGoal(goal);
 ac.waitForResult();
 if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
        ROS_INFO("Hooray, I moved in the -y direction!");
}

void chatterCallback3(const std_msgs::String::ConstPtr& msg)
{
 ROS_INFO("I will now move in the [%s] Direction", msg->data.c_str());
 MoveBaseClient ac("move_base", true);
 while(!ac.waitForServer(ros::Duration(5.0))){
  ROS_INFO("Check your map file path if More than 5 Seconds!");
```

```
  }
  move_base_msgs::MoveBaseGoal goal;
  goal.target_pose.header.frame_id = "base_link";
  goal.target_pose.header.stamp = ros::Time::now();
  goal.target_pose.pose.position.x = .5;
  ac.sendGoal(goal);
  ac.waitForResult();
  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
          ROS_INFO("Hooray, I moved in the +x Direction!");
}

void chatterCallback4(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I will now move in the [%s] direction", msg->data.c_str());
  MoveBaseClient ac("move_base", true);
  while(!ac.waitForServer(ros::Duration(5.0))){
    ROS_INFO("Check your map file path if More than 5 Seconds!");
  }
  move_base_msgs::MoveBaseGoal goal;
  goal.target_pose.header.frame_id = "base_link";
  goal.target_pose.header.stamp = ros::Time::now();
  goal.target_pose.pose.position.x = -.5;
  goal.target_pose.pose.orientation.w = .5;
  ac.sendGoal(goal);
  ac.waitForResult();
  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED)
          ROS_INFO("Hooray, I moved in the -x Direction!");
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("chatter", 1, chatterCallback);
  ros::Subscriber sub2 = n.subscribe("chatter2", 1, chatterCallback2);
  ros::Subscriber sub3 = n.subscribe("chatter3", 1, chatterCallback3);
  ros::Subscriber sub4 = n.subscribe("chatter4", 1, chatterCallback4);
  ros::spin();

  return 0;
}
```

Batter & Auto-Dock Code

```
#include <ros/ros.h>
#include <move_base_msgs/MoveBaseAction.h>
#include <actionlib/client/simple_action_client.h>
#include <std_msgs/Empty.h>
#include <kobuki_msgs/Led.h>
#include <stdio.h>
/***  Program to put robot to bed via coordinates and auto-dock  **/
typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;
typedef ros::Publisher blink_publisher;
int main(int argc, char** argv){
```

```
int flag = 0; // to check for auto-docking success or failure
//Initialize Program
  ros::init(argc, argv, "go_to_bed");
  MoveBaseClient ac("move_base", true);
  //wait for the action server to come up and check map file is loading
  while(!ac.waitForServer(ros::Duration(5.0))){
    ROS_INFO("Waiting for the move_base action server to come up. Check your map file path!!!");
  }
  move_base_msgs::MoveBaseGoal goal;
  /* we'll send a goal to the robot to go to bed.  The x and y position coordinates represent  A location
close enough to the charging dock for the ir sensors to work  */
  goal.target_pose.header.frame_id = "map";
  goal.target_pose.header.stamp = ros::Time::now();
  goal.target_pose.pose.position.x = -0.919494390488;
  goal.target_pose.pose.position.y =  0.334971427917;
  goal.target_pose.pose.position.z =  0.0;
  goal.target_pose.pose.orientation.x = 0.0;
  goal.target_pose.pose.orientation.y = 0.0;
  goal.target_pose.pose.orientation.z = 0.00624722190278;
  goal.target_pose.pose.orientation.w = 0.999980485919;
 // Print to terminal command has been issued
  ROS_INFO("Shushbot!  Go to bed!");
// Send the goal position to Kobuki base
 ac.sendGoal(goal);
// Wait for confirmation goal was reached
  ac.waitForResult();
// If goal was reached activate auto docking program
  if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
// Print to screen showing auto docking has been activated
    ROS_INFO("Shushbot is heading to bed!");
    flag = system("roslaunch kobuki_auto_docking activate.launch");
    if(flag){
// Print to screen that auto-docking was successful
    }
    else{
// Auto-docking was NOT successful. Need to physically check robot.
    }
  }
  else{
// Goal was never reached. Need to physically check robot.
    ROS_INFO("The Shushbot failed to find his bed");
  }
  return 0;
}
```

Starter-Up Program

```
#!/bin/bash
# Turtlebot Startup Script

echo "Launch Minimal"
xterm -hold -e roslaunch turtlebot_bringup minimal.launch &
sleep 15
```

```
echo "Launch Autodocking Algorithm"
xterm -hold -e roslaunch kobuki_auto_docking minimal.launch &#--screen

sleep 5

echo "Load Map"
xterm -hold -e roslaunch turtlebot_navigation amcl_demo.launch map_file:=/home/turtlebot/Desktop/MyMap4.yaml
&

sleep 5

echo "Load Battery Program"
xterm -hold -e rosrun turtle batListen &

sleep 15

echo "rosrun USB"
xterm -hold -e rosrun rosserial_python serial_node.py _port:=/dev/ttyACM0 &

sleep 15

echo "roschatter"
xterm -hold -e rostopic echo chatter &
sleep 2
xterm -hold -e rostopic echo chatter2 &
sleep 2
xterm -hold -e rostopic echo chatter3 &
sleep 2
xterm -hold -e rostopic echo chatter4 &
```

Sound Sensor Program

Arduino Sensor Program

```
/*
 * rosserial Code for Shushbot moving
 * Prints "Move in xyz-direction"
 */
```

```
//#define USE_USBCON
#include <ros.h>
#include <std_msgs/String.h>

ros::NodeHandle  nh;

std_msgs::String str_msg;
ros::Publisher chatter("chatter", &str_msg); //This is for Stacy's program to turn & move in the direction of the noise
ros::Publisher chatter2("chatter2", &str_msg);// Turn 180 means the sound is behind the robot
ros::Publisher chatter3("chatter3", &str_msg);
ros::Publisher chatter4("chatter4", &str_msg);
ros::Publisher chatter5("chatter5", &str_msg);
ros::Publisher chatter6("chatter5", &str_msg);


//Sensor1=front sensor=LED2
//Sensor2=rear sensor=LED3
//Sensor3=right sensor=LED4
//Sensor4=left sensor=LED5

int i;

const int sampleWindow = 1000; // Sample window width in mS (50 mS = 20Hz)
unsigned int sample1;
unsigned int sample2;
unsigned int sample3;
unsigned int sample4;

double SensorValue1=0.0; //Set the 4 sensor readings to null
double SensorValue2=0.0;
double SensorValue3=0.0;
double SensorValue4=0.0;

//int M1=A0; //Naming the analog inputs to microphone sensors
//int M2=A2;
//int M3=A4;
//int M4=A6;

double Threshmin = 1.5;// This value is dependent on the sensitivity of the sensor SET TO 3 for normal talking
voice

double Threshmax = 2.00; // This value is dependent on the sensivitiy of the sensor

char Forward[] = "Forward";//Sensor 1
char Turn_180[] = "Turn_180";// Sensor 2
char Right[] = "Right";//Sensor 3
char Left[] = "Left";//Sensor 4
char SHUSH[] = "SHUSH";//Quiet Program
char Scan[] = "Scan"; //Scan Again


void setup()
{
  nh.initNode();
  nh.advertise(chatter);// THe advertise function is what allows Stacy's program to listen for this specific command
  nh.advertise(chatter2);
  nh.advertise(chatter3);
  nh.advertise(chatter4);
```

```
  nh.advertise(chatter5);

  //Serial.begin(9600);// Set the Baud rate for the Arduino
}

void loop()
{
  unsigned long start= millis(); // Start of sample window
  unsigned int peakToPeak1 = 0; // peak-to-peak level
  unsigned int peakToPeak2 = 0; // peak-to-peak level
  unsigned int peakToPeak3 = 0; // peak-to-peak level
  unsigned int peakToPeak4 = 0; // peak-to-peak level
  unsigned int signalMax = 0;
  unsigned int signalMin = 1024;
  unsigned int signalMax1 = 0;
  unsigned int signalMin1 = 1024;
  unsigned int signalMax2 = 0;
  unsigned int signalMin2 = 1024;
  unsigned int signalMax3 = 0;
  unsigned int signalMin3 = 1024;
  unsigned int signalMax4 = 0;
  unsigned int signalMin4 = 1024;

  while (millis() - start < sampleWindow)
  {
    sample1 = analogRead(A0);
    sample2 = analogRead(A2);
    sample3 = analogRead(A4);
    sample4 = analogRead(A6);

//    for (i=0;i<3;i++)
//    {
//      sample1 = sample1 + analogRead(A0); //Read the value of noise
//      sample2 = sample2 + analogRead(A2);
//      sample3 = sample3 + analogRead(A4);
//      sample4 = sample4 + analogRead(A6);
//
//      delay(500);
//    }
    // Display the values of each of the sensor readings for the serial monitor

       //Serial.println(sample1);
    //Serial.println(sample2);
    //Serial.println(sample3);
    //Serial.println(sample4);
//
//    sample1 = sample1/3;
//    sample2 = sample2/3;
//    sample3 = sample3/3;
//    sample4 = sample4/3;

//    Serial.println(sample1);
//    Serial.println(sample2);
//    Serial.println(sample3);
//    Serial.println(sample4);
//    Serial.println("_________________");
//    delay(1000);
```

```
if (sample1 < 1024) // toss out spurious readings
  {
   if (sample1 > signalMax1)
      {
        signalMax1 = sample1; // save just the max levels
        //Serial.println(sample1);
      }

   else if (sample1 < signalMin1)
      {
        signalMin1 = sample1; // save just the min levels
        //Serial.println(sample1);
      }
  }

if (sample2 < 1024) // toss out spurious readings
  {
   if (sample2 > signalMax2)
      {
        signalMax2 = sample2; // save just the max levels
        //Serial.println(sample2);
      }

   else if (sample2 < signalMin2)
      {
        signalMin2 = sample2; // save just the min levels
      }
  }

if (sample2 < 1024) // toss out spurious readings
  {
   if (sample2 > signalMax2)
      {
        signalMax2 = sample2; // save just the max levels
        //Serial.println(sample2);
      }

   else if (sample2 < signalMin2)
      {
        signalMin2 = sample2; // save just the min levels
        //Serial.println(sample2);
      }
  }

if (sample3 < 1024) // toss out spurious readings
  {
   if (sample3 > signalMax3)
      {
        signalMax3 = sample3; // save just the max levels
      }

   else if (sample3 < signalMin3)
      {
        signalMin3 = sample3; // save just the min levels
      }
  }

if (sample4 < 1024) // toss out spurious readings
```

```
    {
     if (sample4 > signalMax4)
       {
         signalMax4 = sample4; // save just the max levels
       }

     else if (sample4 < signalMin4)
       {
         signalMin4 = sample4; // save just the min levels
       }
    }
   }

   peakToPeak1 = signalMax1 - signalMin1; // max - min = peak-peak amplitude
   peakToPeak2 = signalMax2 - signalMin2; // max - min = peak-peak amplitude
   peakToPeak3 = signalMax3 - signalMin3; // max - min = peak-peak amplitude
   peakToPeak4 = signalMax4 - signalMin4; // max - min = peak-peak amplitude

   SensorValue1 = (peakToPeak1 * 50.0) / 1024; // convert to volts NOTE: the 50V works as the amplification
   SensorValue2 = (peakToPeak2 * 50.0) / 1024; // convert to volts NOTE: the 50V works as the amplification
   SensorValue3 = (peakToPeak3 * 55.0) / 1024; // convert to volts NOTE: the 50V works as the amplification
   SensorValue4 = (peakToPeak4 * 45.0) / 1024; // convert to volts NOTE: the 50V works as the amplification
//
//   Serial.println("_________________");
//   Serial.print("Forward ");
//   Serial.println(SensorValue1);
//   Serial.print("Back ");
//   Serial.println(SensorValue2);
//   Serial.print("Right ");
//   Serial.println(SensorValue3);
//   Serial.print("Left ");
//   Serial.println(SensorValue4);

   //Start the noise detection algorithim
   //if (SensorValue1||SensorValue2||SensorValue3||SensorValue4>=Threshmin)
   //{
    //The algorithim will compare the different sensor inputs and turn in the
    //diretion of the highest threshold sensor

    if (SensorValue2>Threshmin)
    {
     if (SensorValue2>SensorValue1 && SensorValue2>SensorValue3 && SensorValue2>SensorValue4)
     {
      str_msg.data = Turn_180;
      chatter2.publish( &str_msg );

      Serial.println("I Turn 180");
      Serial.println(SensorValue2);
      delay(4000);
      //Check if it has reached its max Thus activating SHUSH Func.
      if(SensorValue2>=Threshmax)
      {
       str_msg.data = SHUSH;
       chatter5.publish( &str_msg );

       Serial.println("SHUT UP HUMAN");//The Stop of Stacys' program will happen here and activate Robertos'
SHUHSH Func.
       delay(1000);
```

```
    }
    else{}
  }
  else
  {
   SensorValue2=0.00;
  }
}
else
{
 SensorValue2=0.00;
}

if (SensorValue3>Threshmin)
{
 if(SensorValue3>SensorValue1 && SensorValue3>SensorValue2 && SensorValue3>SensorValue4)
 {
    str_msg.data = Right;
    chatter3.publish( &str_msg );

    Serial.println("I Move Right");
    Serial.println(SensorValue3);
    delay(4000);
   //Check if it has reached its max Thus activating SHUSH Func.
   if(SensorValue3>=Threshmax)
     {
      str_msg.data = SHUSH;
      chatter5.publish( &str_msg );

      Serial.println("SHUT UP HUMAN");
      delay(1000);
     }
   else{}
 }
 else
 {
  SensorValue3=0.00;
 }
}

else
{
 SensorValue3=0.00;
}
if (SensorValue4>Threshmin)
{
 if (SensorValue4>SensorValue1 && SensorValue4>SensorValue2 && SensorValue4>SensorValue3)
 {
  str_msg.data = Left;
  chatter4.publish( &str_msg );

  Serial.println("I Move Left");
  Serial.println(SensorValue4);
  delay(4000);
  //Check if it has reached its max Thus activating SHUSH Func.
  if(SensorValue4>=Threshmax)
  {
    str_msg.data = SHUSH;
```

```
          chatter5.publish( &str_msg );

          Serial.println("SHUT UP HUMAN");
          delay(1000);
        }
        else{}
       }
       else
       {
        SensorValue4=0.00;
       }
      }

      else
      {
       SensorValue4=0.00;
      }

      if (SensorValue1>=Threshmin)
      {
       if(SensorValue1>SensorValue2 && SensorValue1>SensorValue3 && SensorValue1>SensorValue4)
       {
        str_msg.data = Forward;
        chatter.publish( &str_msg );

        Serial.println("I Move Forward");
        Serial.println(SensorValue1);
        delay(4000);
        //Check if it has reached its max Thus activating SHUSH Func.
        if(SensorValue1>=Threshmax)
        {
          str_msg.data = SHUSH;
          chatter5.publish( &str_msg );

          Serial.println("SHUT UP HUMAN");
          delay(1000);
        }
        else{}
       }
       else
       {
        SensorValue1=0.00;
       }
      }
     else
     {
      SensorValue1=0.00;
     }
//    else
//     {
//      str_msg.data = Scan;
//      chatter6.publish( &str_msg );
//      Serial.print("I wont move");
//      delay(1000);
//     }

     Serial.println("_________________");
     Serial.print("Forward ");
```

```
    Serial.println(SensorValue1);
    Serial.print("Back ");
    Serial.println(SensorValue2);
    Serial.print("Right ");
    Serial.println(SensorValue3);
    Serial.print("Left ");
    Serial.println(SensorValue4);

  nh.spinOnce();
  delay(500);
}
```