Thunderbot's Software Projects

1 Pan/Tilt Head Controller

Description:

A mount for the robot's head that can both pan and tilt (look left/right and up/down) with respect to the torso. It will have to mount to the torso and have cameras and a Kinect mounted to it. **(any other sensors to mount to the head? microphone?)**

You will be given a model and an image of the pan/tilt head and you must write a software controller for it. The image is just to help you visualize the project. The controller should take in pan_tilt messages from our ROS (Robot Operating System) integrated system and use the information in those messages to move the head.

For example, a message stream will come in in which your C++ or Python file will listen for on the topic pan_tilt_topic and it will contain information like:

angular_x
angular_y
angular_z

and you will have to make the head move with respect to those commands. If the command only has information to make it move 30 degrees left, the head should stop its old movement and start moving 30 degrees left. It will always adjust to the current information stream it is receiving.

You will also need to create the message type if there is not a similar message type already defined in the ROS libraries.

Requirements:

- Be able to pan at least 90 degrees
- Be able to tilt at least 90 degrees
- Repeatable, smooth movement
- C++ or Python knowledge
- Ability to learn the ROS api (It's not bad, I can point you to tutorials and I'm here if you need help)

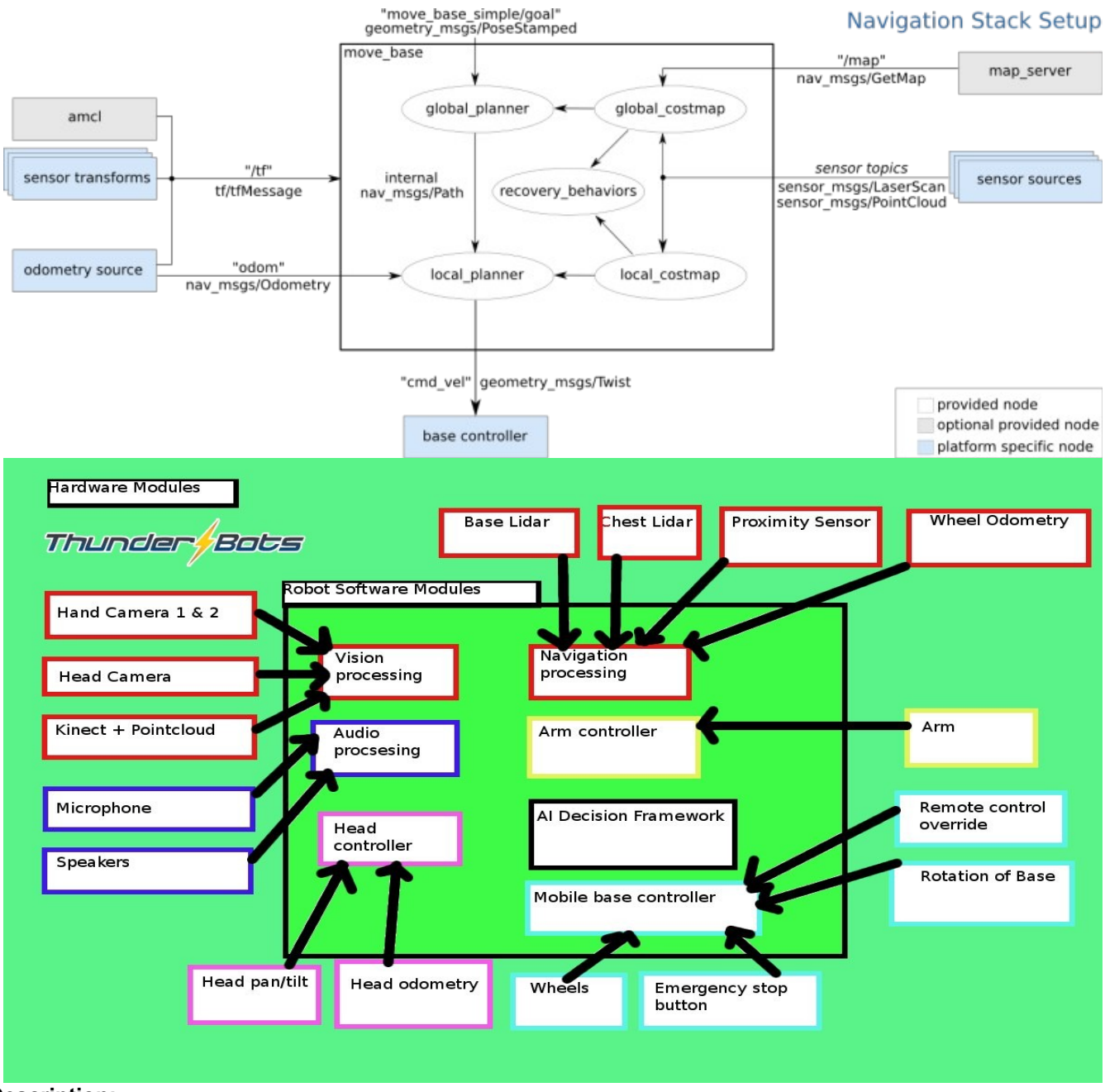
Inputs:

- pan_tilt_msg containing the information for the module
- odom_msg containing the odometry of the head

Outputs:

- Must contact the hardware and make it move
- The current position to the system, eg pan_tilt_position_topic

2 ROS Navigation stack and Drivetrain integration



Description:

The drivetrain software has been completed by Michael Moritsugu (Undergrad, Computer Science) and Justin (Undergrad, Engineering Physics) (sorry, not sure of his last name). Our job is to now to get the drivetrain accepting goals from software and converting them into wheel speeds. We have to create the navigation stack and get it publishing data to our mobile base controller. In the green image above, this will happen in our Navigation processing unit.

ROS Tutorial: http://wiki.ros.org/navigation/Tutorials

Here are some things that the navigation stack requires to run, as per the first image.

1. Sensor transforms
2. AMCL (Monte Carlo Localization)
3. Odometry of the robot/wheels with respect to our defined center point
4. A base controller (written already by Mike and Justin)
5. Sensor topics
6. A map of the world (created with use of a oscillating lidar)

And as such, we will break them down into their respective pairings to make it easier for the owners of the projects to complete. Once these smaller projects are complete then we can write a roslaunch file that starts them all up and starts the navigation stack.

2.1 Sensor topics and Sensor Transforms

Description:

List of our sensors:

1. Hand camera 1 (Not Purchased) (old webcoms, prototypes)
2. Hand camera 2 (Not Purchased) (old webcams, prototypes)
3. Wheel odometry
4. Head odometry (Not Purchased) (being built last)
5. Emergency stop button (Not Purchased) (Alannah to purchase from mcmaster tonight)
6. Rotation odometry of base
7. Remote control override (Not Purchased) (2 wireless xbox controllers)
8. Microphone (Not Purchased)
9. Speakers (Not Purchased) (Alannah, detailed schedule on sensors)
10. Kinect
11. Base LIDAR (Not Purchased)
12. Head Camera (Not Purchased) (Matt and Logan, Alannah Avigilon head camera)
13. Arm odometry
14. Proximity Sensor (Rory working on it)
15. Chest LIDAR

The robot requires sensors to interact with the outside world. With ROS, it becomes easy to get the sensor information to the rest of our system even if it is composed of multiple computers. The owner of this project must write a module in C++ or Python that initializes all of the sensors in the system and checks/confirms that they are on by receiving data from them. The module must then publish it to the system and do another check that it is in the system by trying to subscribe to these topics.

Once the information is published by the system and everything is tested to make sure it is working, we must publish the sensor transforms. You can learn how to do a sensor transform from the ROS tutorials on it. Ask Devon if you need guidance or direction towards these tutorials. Once the transforms are published to the system, you must have it check with an automated test and confirm that they are in fact being transformed

correctly. If this is done we can move onto our next sub module.

2.2 Map of the world and AMCL

Description:

Since our sensors are working healthily as of 2.1, we can now use the LIDAR to create a map of the world. This will be done by turning the lidar on and creating a .bag file of the information. We will have to do this again for the competition arena. But for now, the goal is to create a .bag file of the top floor of the EDC where the software team meets. Once that is done, complete the rest of the navigation stack tutorials to figure out all the small files the nav stack requires to run properly. There will be a very good tutorial explaining the whys, hows, and the how to dos. Ask me for guidance and reference and I will point you towards the tutorials.

Requirements 1:

.bag file containing the information of the map of top floor of edc.
Should be able to create a map of any room on demand (eg, write a script that starts the lidar and starts making a map)

Requirements 2:

Follow this tutorial to get AMCL up and running. http://wiki.ros.org/navigation/Tutorials/RobotSetup

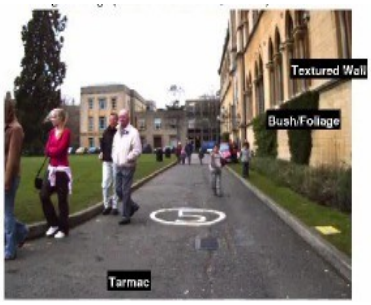
2.3 Navigation stack completion

Description:

Follow this tutorial, as we have all components now. http://wiki.ros.org/navigation/Tutorials/RobotSetup

3 Scene Recognition

Description:

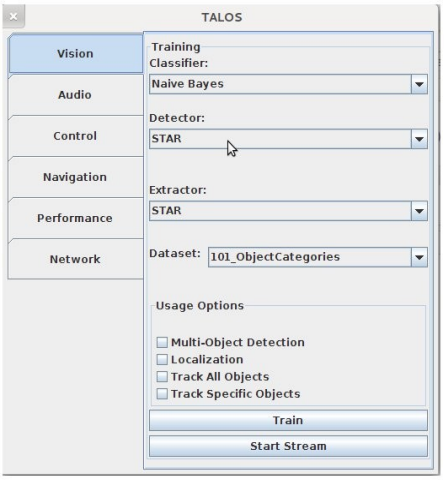


The purpose is to classify whether we're in a living room, kitchen, inside, or outside. The project owner will have to search for research papers that successfully implement the classification of scenes and extraction of scenes or find libraries that enable this to happen. Your job is to recreate what they had created in the paper with similar metrics. Aim for the highest performance research, and the most cutting edge.

4 Onboard Visual Interface

Description:

The onboard visual interface was started by myself (Devon) and it is written in Java. The purpose of the visual interface was to provide a way for other project goers, or people who have little knowledge of command line interactions to be able to communite with the robot. The interface is meant to encapsulate the entire back-end and control all of the hyperparameters of the system. There are tabs that allow the user to change what they see and thus what they can control.



There must be a start button and a stop button, the start button will initialize everything on the robot. I will define what each button and tab must do and/or contain, below.

The creator may decide where to put the buttons. Our aim is functionality and a bit of organization, but don't over design or try to make it too pretty as only engineers will be using it. There will be a dropbox link to where the source code is put (bottom of this section) and you can javac it yourself and run it yourself. There may be absolute pathing errors, if you find some report them to me and I will fix them.

General Buttons:

The general buttons should be in sight at all times, regardless of what tab you're in.

Start

The start button needs to initialize all of the sensors on the robot, print out a performance check, print out a status of each module(running, not running), print out a status of each sensor, (eg, running, not running), a list of computers that are on the network (client, server of robot) and relevant to the robot. You will need to spawn multiple threads to do this or else the rest of the gui won't work while it's starting up.

Stop

The stop button needs to properly close and save the status of the gui and the robot. This includes

properly notifying the user that all the sensors have been shut off and letting them know that the system is now off.

Emergency stop

The emergency stop should immediately cut all power in the system, akin to the hardware emergency stop button. It should keep the gui open, but disable all functions until the user clicks “Everything is ok” which will be prompted to the user.

Tabs:

Vision

The vision tab should say what model the system is using for classification, scene recognition, and so forth. It should have a list of all the objects currently able to be recognized by the system. The current classifier, detector, and extractor things should go away and a new design will be implemented. The dataset tab should also go away and be replaced with something like “Model name” and “Training algorithm”, “Optimization technique”, ... etc. Ask me for more details later.

Vision buttons

The buttons in the vision tab should be train, show weights, and print_monitor, show weights should show the parameters in the model visually and the print_monitor should show the statistics of the model. The train button should train the model on its respective data on a seperate thread to prevent freezing in the system.

Audio

The audio tab is designed for the user to be able to input text that the robot must synthesize, and record the sound it is hearing. Thus we have a couple buttons. It should also have a slider to adjust the output volumes and input microphone volumes. It must also have a description of what model it is trained on, eg GMM, DNN, RNN, HMM and its specific parameters that make it work.

Audio buttons:

Record
Volume Sliders
Mute/Sound on/off
Microphone on/off
Text synthesizer field (eg input into field, the robot says it)

Performance

Performance should include statistics like the current recognition rate of the vision and audio modules, the temperature of the system, the wheel speed, the direction, the head direction and head turning speed. Other statistics that are important that you can think of should also be added.

Network

The network tab should contain information such as the list of computers currently connected to the

robotic system, the sensor's status, modules status, whether things are connected, connecting, or disconnected, and if they are connected, their latencies. It should also provide a connection code for the client that will be made later in an Android application. Not only that, it needs to be able to have the IP address of the robot (both lan and outside IP. Eg 192.168.0.1 vs 154.20.158.165)

Network buttons:

It should have a start server button when pressed makes the robot available for connection from the client with the code. When the button is stopped the server should cut connection from the client. There should only be allowed one client at a time.

Dropbox Link:
https://www.dropbox.com/home/Public/ThunderbotsHomeGUI

5 Arm Controller

Description:

A node to interface between ROS and the nodes in the arm.

6 Bloop

Description:

Blippity blop bloop

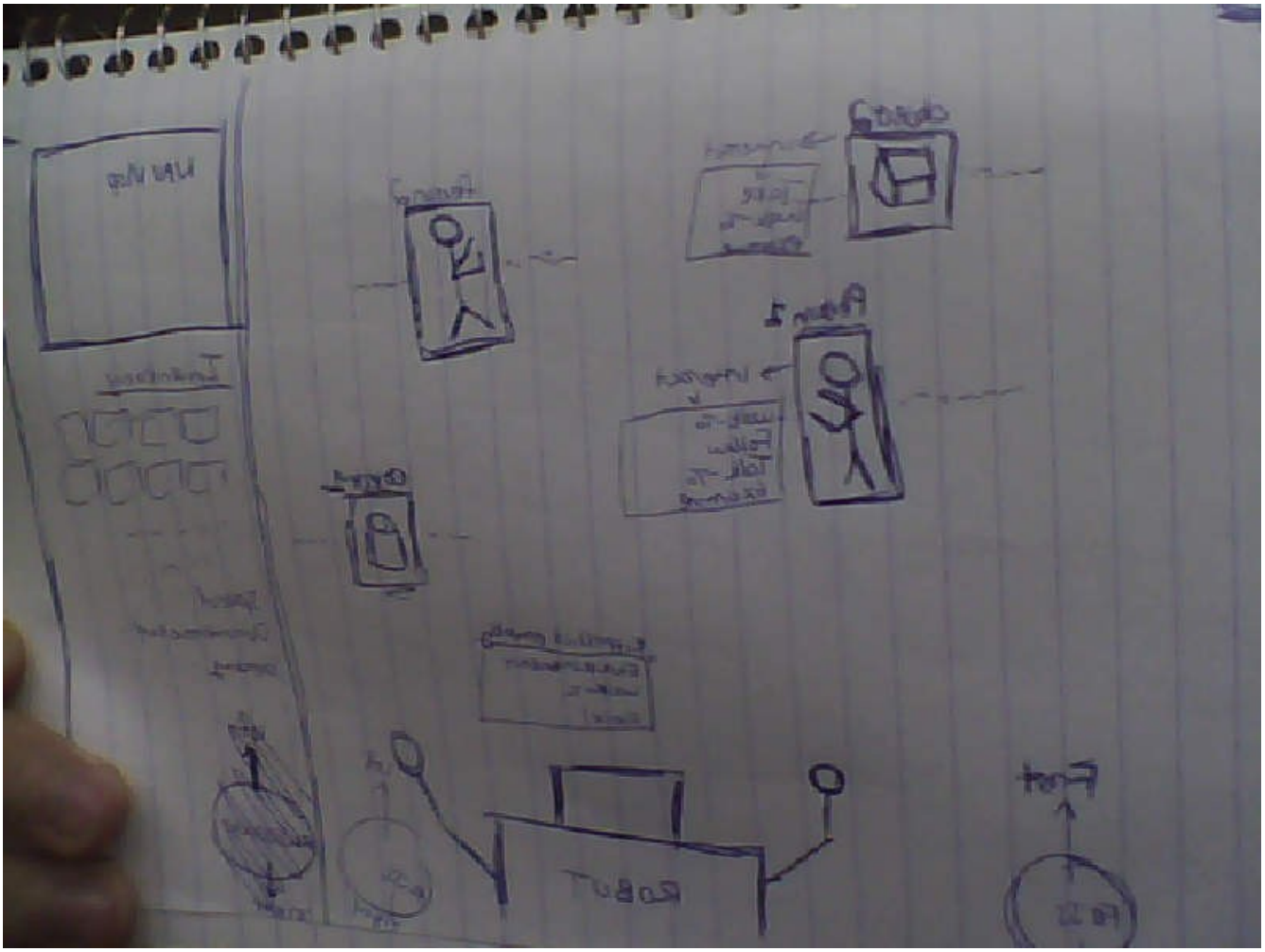
7 Mobile Base Controller

Android Application Specifications

Talos, Thunderbots@Home, UBC

PURPOSE:

The purpose of an android application for the Talos robot is to provide a means of controlling the robot and view its sensor data in a user friendly environment. Currently, this is how I would like the user interface to look:



CONTROL OF ROBOT

MOVEMENT

The two circular things at the bottom to the left and right of the robot are the joysticks. One joystick will control front and backward movement whilst the other one would control turning. (Just like in a playstation controller). The robot would be a static image overlaid onto the image. To move its arms, I have two solutions that may be possible. One of which is cooler. I would like to have the camera of the android process something like the Kinect does (if possible) and move the arms by copying what the human does with its arms. However, this is completely stupid because the human will have his hands on the joysticks and therefore this will be impossible. Keep reading.

(The real solution) The robot will moves its arms by having the user select its joints and manoeuvring them. For example, to move the hand to a specific location we could select the hand node and drag it and the robot's hand should move there.

INTERACTION

The robot will interact with the world by having bounding boxes drawn around each person and each object it identifies. On the outside of the box will show the name of the class or instance of the object so the user will know. To interact, the user may right click on any of the boxes to receive a menu that will show what the robot can do with said object. In the case of humans, for example, the robot would have in the menu: talk, walk-to, follow, and examine. Each of the interaction options does exactly what the word states.

INFORMATION

The user interface will show what objects the robot currently has on it (in its inventory) and a map of the surrounding area. It will also attempt to identify the environment that it is in and show it beside the map. If it determines that the objects in the scene make up a kitchen, then it will show “Environment: Kitchen”.

STATUS

The status will be on the side and contain things like wheel speed, velocity, its current actions, current goal points, and network status.

8 Xbox Controller for the Robot's Base (Remote Control Override and Emergency stop)

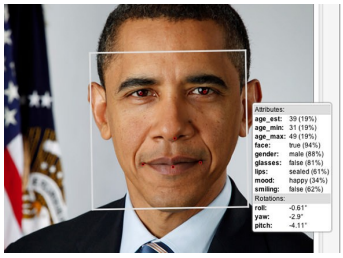
Description:

The owner of the project will create a node that listens to the wireless receiver of the xbox controller. The owner will take the xbox controller's commands and turn them into base commands for the robot and send them to the base. The xbox controller should always override the commands given by the AI and the emergency stop button should also be implemented via this method. (EG, press X and robot stops/turns off)

To do this you will create an executable cpp/py file that contacts the usb of the xbox controller and maybe find a xbox controller api that you can use. You should import that xbox controller api to interpret the commands and import the ros libraries to be able to contact the rosservice on the mobile base. You should send movement messages to the base of our system when you interpret the commands from the xbox api send it to the base and the base will handle it from there.

9 Facial Recognition

Description:



The owner of the project will find an opensource facial recognition api and possibly use the one provided by ROS. You can search google to find it. You will need to provide a rosservice to the system called “get_detected_faces” that returns a list of all the names of the faces detected. There should also be a topic called “get_picture_of_face” which returns the picture of the face given the name. The last service provided should be something called “get_names_with_pictures” which returns pictures of the faces and the names corresponding to the picture.

This is required in the competition for a multitude of tasks. The robot must use it for recognizing persons it has to serve and persons it must follow. It is also a piece in a larger task which will be used for entire person recognition (Face + body).

10 Gesture Recognition

Description:

The owner of the project must create a module in python or c++ that listens to the cameras from the head and recognizes a gesture. You might want to look toward the kinect api for recognizing a pointing motion of the human. You host a rosservice in the module called “get_point_direction” that returns the direction of the pointed finger relative to the robot.

11 Speech Recognition & Speech Synthesis & Natural Language Processing

Description:

This is a two part project. The speech recognition part given as input through the microphone, and the synthesis part given out of the speakers. We will break this up into two sections, 11.1, 11.2. Natural language processing can occur after both of these are made available to our system, 11.3. You can see what the microphones look like on top of the Cosero robot to the right.

11.1 Speech Recognition

Description:

The owner of this project must create a service that listens to the microphone input, and add it to the ros system. The module should convert the speech into text and publish it to the system on a topic called “recognized_text” and anything that subscribes will be able to see the latest sentence heard by the system. The module will also have a data structure that keeps sentences for up to 30 seconds.

11.2 Speech Synthesis

Description:

The owner of the project must create a module that hosts a rosservice called “text_to_speech” and takes text as a parameter and inside the module it should send the sound using some API to the speakers. I'd suggest finding an online opensource solution as there is no point to reinvent the wheel here.

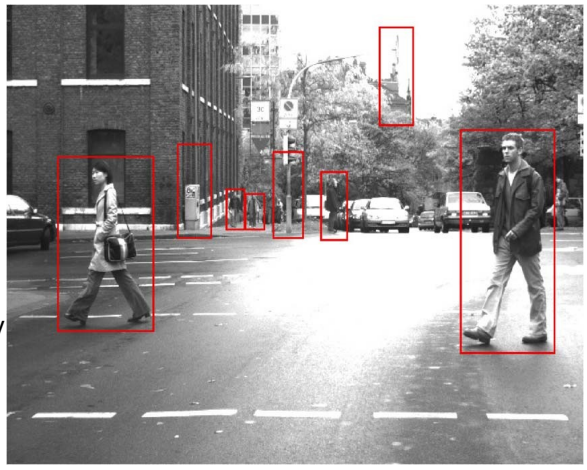
11.3 Natural Language Processing:

Description:

The owner of the project should create a module that contacts the speech recognition module and uses its text and its text history to understand the commands, persons, and objects involved in the speech. There should be commands programmed for each time it recognizes a responsible command, but that can come later, at the AI stage. Right now we just need topic published called “command_heard”, which will require a defined message in the ros system that includes the parameters which will be published in the command_heard message.

12 Person recognition & Tracking

Description:



The owner of the project must read some research on how to do person recognition and find out how to do it within our system. They also must implement some tracking system that is able to draw a box around a human and get their position while moving. The module should have a rosservice that takes in as a parameter a picture of a persons face + body and maybe some other things as well and it should find that person and draw a box around them that is highlighted. The other persons in the image or video should be tracked as well but not with the same box colour.

13 Obstacle Detection & Avoidance

Description:

The owner of the project must do something with either the kinect, lidars, cameras, or a combination of all to determine what is an object in the scene. They must then publish the location of these objects out to the system in a topic called “obstacles_in_scene” which provides a list of all the locations of obstacles. This will be useful for the nav stack or for other things.

ROS apparently has something already so take a look at this first.

http://wiki.ros.org/turtlebot_navigation/Tutorials/Build%20a%20map%20with%20SLAM