Nathan, Bri, Peter

Responding to Voice Commands

Abstract:

The goal of this project was to improve robot human interaction through the use of voice commands as well as improve user understanding of the robot's state. Our approach in improving these issues was to allow the user to give voice commands to the robots instead of having the user type them into a terminal. LED lights were also used to portray the current state of the robot, in order to make it clear what was occurring in the program, especially to people with little to no programming experience. We achieved this through the use of the voice recognition API, Sphinx, created by Carnegie Mellon and Arduino, to communicate to the LEDs from the computer.We achieved having the robot make movements based on a set of predetermined voice commands (forward, backward, turn left, turn right, dance, party) and the led states (a wrapping green/yellow for movement, blinking red for not understanding the command given, and interesting color patterns for party and dance).

Introduction:

One of the current pressing issues with autonomous intelligence is the need for human input for robots to operate. This is often done through the use of terminal commands or GUI interface, which makes it rather inaccessible for people not familiar with these tools. In order to help overcome this problem there are several approaches one could take, such as creating a program to take in all information at once and have the robot try to react to a changing environment, which could be difficult to do since one would need to consider all possible variables. Another way is to have the human user give commands through motion or speech to help avoid unexpected events, while still maintaining an interaction between the human or robot that anyone could use. We chose to do latter with speech along with LEDs to indicate the state of the robot to make it clear to everyone what is occurring at each point while the program is running.

Background:

Carnegie Mellon has a speech API called sphinx which is used for voice recognition. This was created through the use of machine learning and hidden markov models in which they spent an inordinate amount of time figuring out the approximate frequencies of numerous words along with the thresholds the sound could be to accurate. They then ran trials of each word and of words that didn't fit to improve accuracy of the program. Some common issues with speech recognition include the fact that sound decays over distance, so it requires a person to be nearby for it to be more accurate. The Sphinx API also only recognizes a single word at a time in the predetermined dictionary but other APIs such as LISPER, developed by Daniel G. Bobrow and Dennis H. Klatt, actually recognizes whole phrases at a time. These APIs, however, are more dated and have more consistency issues.

Technical Approach

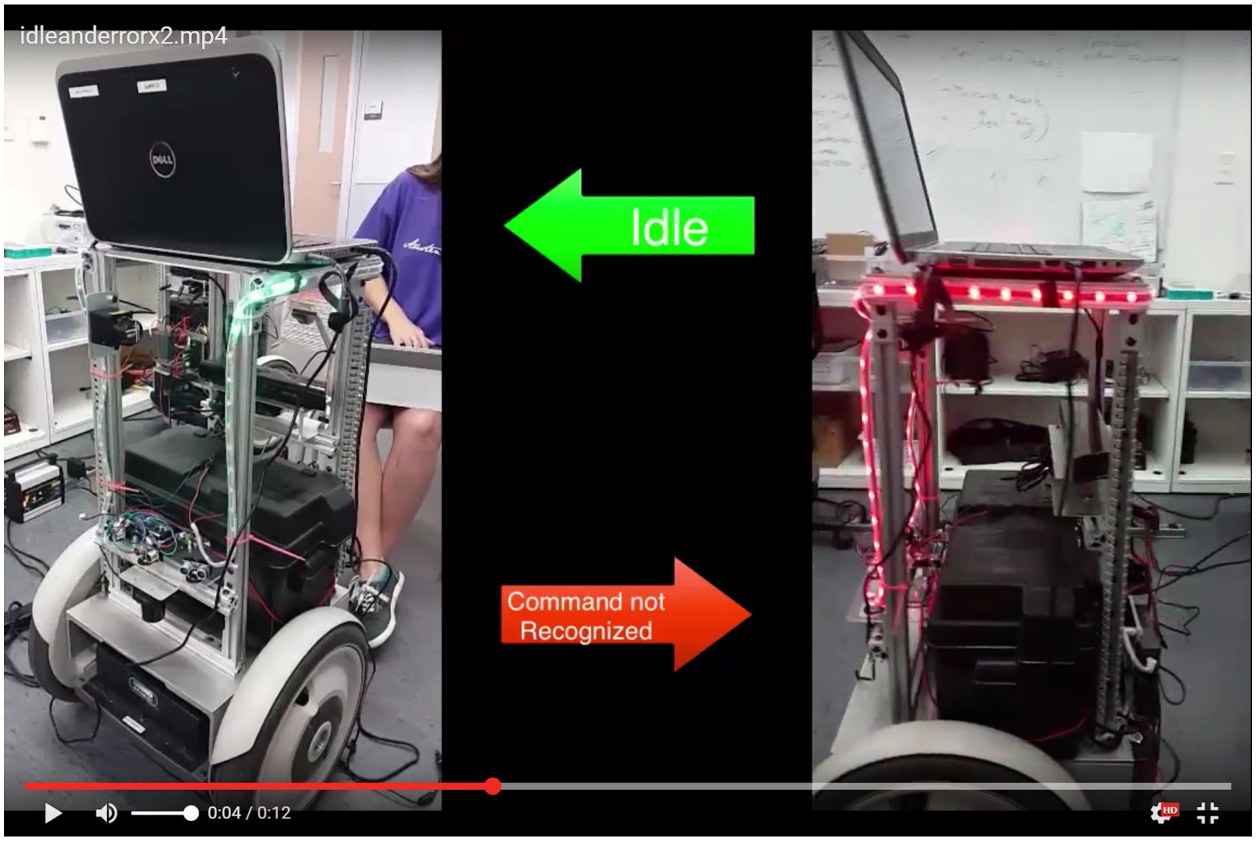
Speech and Movement:

We based our sphinx code off a tutorial made by pi robot and used pocketsphinx to create a predefined dictionary containing forward, backward, turn, left, right, dance, party, come, here. We then compiled this code on Carnegie base knowledge base tool 3 and had our program recognizer start when it hears a noise and stop hearing for that single command after two seconds. So if a command or person talks for longer it is read in as two separate commands. We then had another node subscribe to our recognizer in which if the recognizer noticed a command (forward, backward, dance, party, turn left, turn right, come here - unimplemented with movement) it would modify the premade goal with the appropriate data and add it to a queue collection of movement goals. Our forward and backward commands moved one meter in their respective directions, the turns going ninety degrees of the original position, and dance and party chaining a combination of the prior commands. Also when each of these commands are recognized a string is published to a node indicating the state of the program so that our serial node can and know what state it is in and publish the correct configuration to the LEDs. To be explicit a wrapping green for all movement commands, a blinking red for when the recognizer did not recognize a command, and an interesting pattern for dance and party specified above. We created global quaternions and modified the message data of move base actions to speed up our speech callback function that processed the voice command strings. We had a while loop that continuously checked if the goal queue contained any goals, on which it would pop it off the queue, send it through the simple action client to the base, and wait until the destination had been reached before looping again.

Lights:

The LED light strip served the purpose of expressing the state of the machine in a visual manner. The “state” of the machine depended on which movements and calculations, or lackthereof, were taking place. Because of this, the node which controlled the movement of the robot had to simultaneously communicate with the node that controlled the lights. The first level of this process was a node that opened a serial connection. This node used rosserial as well as a subscriber to the node that controls the movements. The subscriber would read in a string representing the state of the node (i.e. “moving,” “idle,” etc.) and write the corresponding command over the serial connection. Each command sent over began with a char representing the type of action the lights should perform, followed by 3 integers representing the red, green, and blue values of the color in which the action would be carried out. The actions included a ‘w’ for wrap, a ‘b’ for blink, a ‘d’ for dance, and a ‘p’ for party. The node that was uploaded to the Arduino board would then read these commands over the serial connection and behave accordingly. The wrap command would cause five lights to “chase” each other around the strip, using two for loops with a delay to change the array of LED colors and write to the strip. The color was determined by the commands read in over the serial connection. This was used for the idle stage, during which the LED strip would wrap in green, as well as the moving stage, during which the LED strip would wrap in yellow. The blink command was used in the wrong stage, given when the robot does not recognize the command given. This was done by alternating between turning all the lights to red and turning them all off with a delay in between. The dance command generated two random colors and changed every second LED light to the first color, and every remaining one to the second color. The command was called in a continuous loop, so the colors were re-randomized every few milliseconds. The final command was the party command. This chose a random light in the strip to change to a randomly generated color. This also ran in a continuous loop with a delay in between so that all the lights would continue to change as long as no other command was received.

Experiments:



The idle command is shown to the left, using the wrap command and a 0 250 0 rgb value, shown when the robot is idle and waiting for a command. The wrong command is shown to the right, using the blink command and the 250 0 0 rgb value, shown when a command is not recognized.

The program's success was tested in the final stages by simply giving the voice commands and seeing whether the program responded appropriately. However, along the way the tests were done in smaller pieces. First the voice recognition was tested by printing the input received out onto the screen. If the command matched what was said, then the sphinx implementation was working correctly. The move base client was first tested by sending goals manually and making sure the robot moved accordingly. The lights were tested using the Arduino IDE's serial monitor, which allowed the user to manually input values over the serial connection for the Arduino board to receive. From there, the serial node was made and tested by printing out the commands that were being sent over the serial node. After the nodes were put together in order to work simultaneously and communicate with one another, the values for each still printed out on the screen to help with debugging.

In the end, each node worked nearly perfectly on its own, but things got more complicated when they began communicating with each other. The voice commands and movements worked well together, however the lights did not always portray the correct color. This may have been because there were values constantly being sent over the serial connection and the Arduino board may have gotten mixed up between which values represented commands and which represented red, green, and blue values. Given more time, statements could be printed out to show the current color and command being received in the Arduino and help to debug. The Segbot we used also tended to "get lost" and need to relocalize often, which got in the way of testing. It was often unclear whether the program running was causing the movements, or the localization.

Conclusion and Future Work:

Overall we achieved our goals to a degree of giving the robot the ability to perform actions based on predetermined voice commands and having LEDs clearly display what state our program was in. However, we had some major drawbacks in consistency. The recognizer for Sphinx was not always accurate and required a good microphone and a person to be unrealistically close to achieve accuracy when giving the commands. We could improve this in the future by using a better microphone or using a bluetooth device, which would not only improve the microphone quality but allow the commands to be given at a distance from the robot. Another way to improve this would be to use Google's speech recognition API. However, Google's API only allows a limited amount of pull requests per day, which would make testing difficult. Another drawback we faced was the fact that we used the move base client node to publish goals at a predetermined distance from where the robot's starting point (ex. The robot would move forward one meter if the command forward was given). This proved to be an issue because the robot would not move if it perceived an obstacle in its path and would be stuck on the command for a while till the move base command timed out. We could prevent the robot from getting stuck by creating a modified move base node in which the robot would automatically kill the movement goal if the robot perceives an obstacle. It would occasionally lose its localization and get "lost," requiring the robot to relocalize in order for it to work.

Furthermore, movement commands could be more flexible in allowing the user to have parameters in terms of how far to move or turn, such as "turn left 45 degrees". The dictionary would have to be expanded to include a large range of numbers, but much of the code structure already exists.

Additionally, the ability to chain commands together and give more flexible goals would make voice controlled movement much more useful. Chaining commands would be prefaced with the order, "pause", and post faced with "resume" so that the robot does not take off after the first command. Flexible goals with leniency in obstacles involves the robot moving and stopping in front of an obstacle rather than not moving at all if the goal is placed within the obstacle. This would greatly benefit the general use of voice commands as the user must estimate distances.

See our code at :

https://github.com/nj1407/fri_final_project

Link to folder on drive containing powerpoint and pictures and videos

https://drive.google.com/open?id=0B5rLd24GCsMOdUNCMk0wS1VpZU0

Link to a demo

https://youtu.be/vLhi1pdz0n0

Works Cited

Bob, Daniel G. "A Limited Speech Recognition System." *Institute of Electrical and Electronics Engineers* (1968): n. pag. *Https://www.computer.org/csdl/proceedings/afips/1968/5072/00/50720305.pdf*. Web. 13 May 2016.

Goebel, Patrick. "Pi Robot." *Pi Robot*. N.p., n.d. Web. 13 May 2016.

"Sphinx Python Documentation Editor." *Overview — Sphinx 1.4.1 Documentation*. N.p., n.d. Web. 13 May 2016.