Towards Autonomous Warehousing & Delivery

Team 5 / Samantha Wiener (srw2168), Margaret Qian (myq2000), Neil Chen (nwc2112)

Introduction

Much work has been done recently to automate delivery processes using humanoid robots. Approaches to interfaces between humans and robots still require significant development. We propose two implementations of human-robot interfaces: a speech-reactive interface which enables robot navigation to a static goal, and a color-following interface which enables robot navigation to a dynamic goal.

Related Past Work

Much of the inspiration for our project came from work done at CMU on the CoBot robots and voice-controlled systems, such as Amazon's Alexa, Google Home, and Siri, which provide increasingly popular interfaces for task automation. The CoBot robots perform service tasks such as deliveries between rooms; we wanted our implementation to have similar capabilities. The CoBot robots, uniquely, actively interact with humans to aid in tasks that the robots are physically incapable of, such as pressing elevator buttons or picking up objects. We noticed, however, that the CoBot robots are controlled through a web interface. We believe the ability to remotely send commands to the robot via speech would constitute an improvement to the user interface.

Figure 1 From left to right: Kiva, TORU, Fetch. Each platform supports varying levels of autonomous navigation between points given a 2D map of the environment.

We also drew inspiration from warehouse robotics systems, such as the Kiva robots which populate Amazon warehouses; the Magazino TORU, which automates grabbing and collecting inventory; and various solutions from Fetch Robotics for goods hauling and manipulation. All systems share the capability to identify the shelf in which an item is located, and transport a set of items to humans. We mimic this behavior by specifying unique rooms, or even points within the same room, to which our robot can navigate while carrying items or performing deliveries.

Implementation Details

Web Navigation Interface

We built our web interface with Flask and two lightweight clientside Javascript libraries, Annyang and SpeechKITT. We used Annyang as the framework for parsing voice commands into text and we've configured the speech parser to recognize commands such as "move to the table" or "go home". SpeechKITT easily integrated with Annyang to provide us a simple GUI to control speech recognition and display help text. Accessing our web interface takes the user to a screen that listens for voice commands, displays recognizable locations, an option for inputting custom coordinates (mainly for testing purposes), and a form to add a new location mapping. The benefit of this web interface is that it can be easily accessed via mobile platforms as well, so we are not limited to just a desktop environment.

On the backend, we store each location name and corresponding coordinate in a csv file named "navigation_labels.txt". Each time a user says a voice command, we extract the location name, search the file for a matching label and return the corresponding coordinates. We obtained these coordinates by driving the robot to a location in the lab and querying the `/amcl_pose` topic for the robot's current coordinates. The `amcl` package performs probabilistic localization of the robot, and thus reliably tracks the location of the robot relative to its starting point.

Coordinates are sent to the `move_base` ROS package, which acts as an interface to the ROS navigation stack and thus takes in sensor transforms, odometry, localization information, and a map server to generate a local plan which eventually causes robot motion via the `cmd_vel` topic.

Figure 2 A 2D obstacle map of the testing environment.

For color following, we used OpenCV to identify the largest area of color in the robot's image. We use OpenCV to analyze the blob and look at the color detected in the largest area of the camera. Currently we focus on shades of blue to limit the range, but this could be expanded significantly to detect the most pronounced color in the image, or use other metrics to determine which color or area to track. We load frames from the camera on the robot and pass them into a class which handles mask extraction by color. The masked frames are then analyzed for largest blob by area. The initial area and x-centroid of the largest blob are stored. Once the object is within the 50 pxl of its initial location, the ratio between current area and initial area is calculated. If the ratio is

above a certain threshold, the robot moves backwards. If it is under a certain threshold, the robot moves forward. We made sure the object was in the center before the area calculations (fwd/bwd movement) were performed to make sure the differences in area were because of distance, not because the object was too far to the side to be out of the robot's view.

We all worked together on all aspects of this project, but Neil handled the bulk of integrating our work with ROS, Sam handled most of the vision aspects, and Margaret worked on the code for the web interface.

Results

We've built a bare-bones web interface that integrates all the functionality in our proposal. Below is a screenshot of the interface.

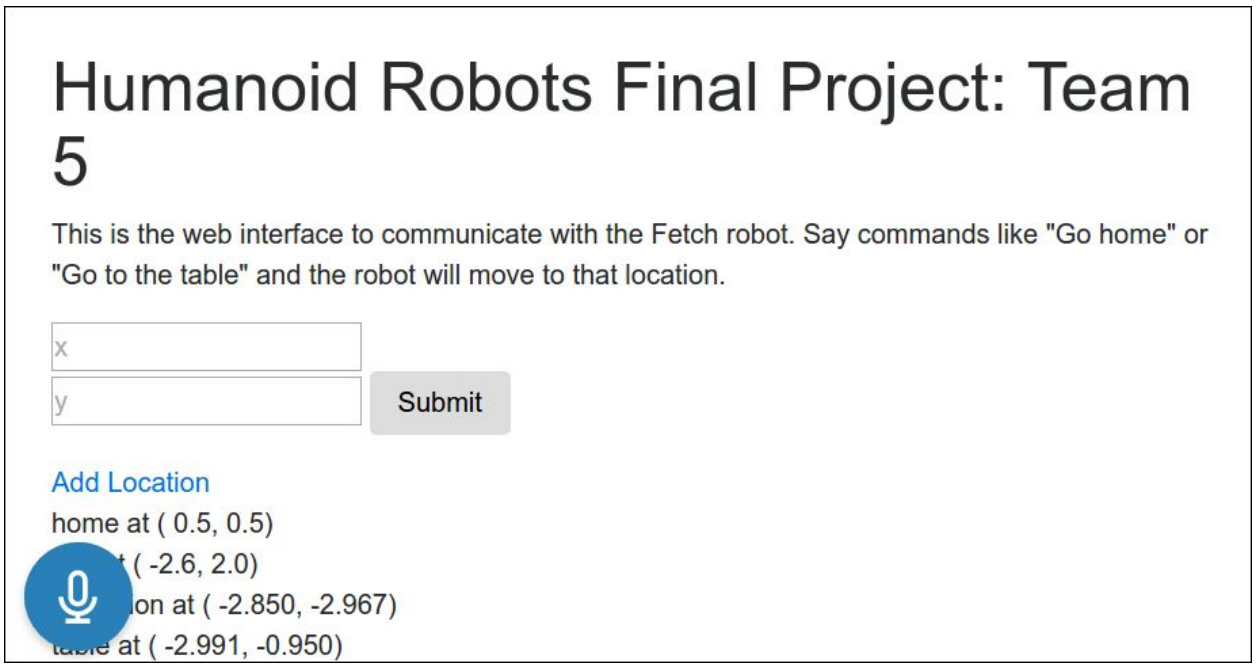

Figure 3 The web interface from which speech commands can be issued.

When the user presses the microphone button and says a command, the Fetch robot moves to the coordinate associated with the location specified in the command. The accuracy of the localization of the Fetch robot in the lab environment provided some challenges. The Fetch robot will often declare that it is stuck even when there are no obstacles in its path. Alongside TAs, we approached this issue by calibrating the robot's position using a vision target and resetting the robot, with varying success.

The Fetch robot also successfully follows a colored object as specified in our proposal. We held a blue bowl to the camera and the Fetch robot was able to follow it at a fixed distance. At the moment, we've defined the range of colors the robot can identify to be just a range of blues, but we can easily expand this to incorporate other colors.

We constructed a pipeline in Python which takes RGB image output by subscribing to the `head_camera/image_rgb` topic. These `sensor_msgs/Image` messages are converted to

OpenCV image `Mat` objects using `cv_bridge`.

The `selector_module` class that handles the mask extraction. A frame is loaded in, and then we use a high and low range of color detection to detect a range of blues in the image frame and determine the area of the blob and its ratio to keep a relative distance as we move the blue object, so that the robot can move forwards/ backwards and left/right to keep a relative distance.

`geometry_msgs/Twist` messages are passed to the Fetch's `cmd_vel` topic, such that robot moves forward or backwards when the target color moves further away or closer; and turns left or right when the target color moves to the left or to the right.

Future Work

Work remains on integrating speech control into robot motion. Since robots like the Fetch are generally expected to operate in controlled environments, such as warehouses or homes, where most debris is static with the exception of humans, we'd like to experiment with fixed beacons from which the robot can more accurately localize itself. We'd also like to experiment with more advanced semantic parsing and natural language paraphrasing, so that speech commands don't need to be structured in a certain format and commands with similar meaning are executed in the same way.

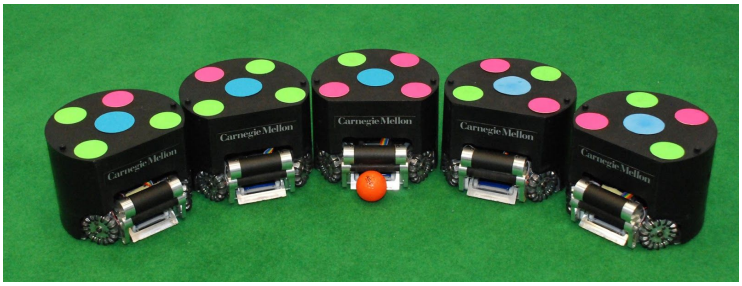

Figure 3 Vision targets arranged in unique patterns, detected by an overhead RGB camera, can enable more robust robot localization. These are RoboCup Small Size League robots.

We also hope to be able to use features of the human body other than color to implement human-following behaviors. Our current configuration requires a human to wear a distinguishable color on their back, possibly aided by a retroreflective surface. By searching for features such as feet or legs, we hope to improve the ability of the Fetch to follow an individual through an environment.

Lessons Learned

The ROS and Fetch platforms make the transition from simulated to experimental environments almost seamless, but real-world constraints strongly limit the effectiveness of our platform. We expected navigation using SLAM in most environments to be a largely solved problem, but found this was untrue during many of our trials with the Fetch in the lab.