Choi, Nathan, Nahian Group Final Project Documentation

Features: (50/70)

- Accumulation and Interpretation sensor data **(10 points)**
- Use of actuators to change the state of the robot **(10 points)**
- A Behavior-based Finite State Automaton with at least four states. **(10 points)**
    - FSMMaster3.py
- At least Four independent ROS Nodes **(10 points)**
    - CliffMonitor, BumpMonitor, LaserMonitor, TargetMonitor
- Demonstration of intelligent behavior - namely ability to robustly operate in dynamic environments **(10 points)**

Implementation: (20/70)

- well written, well documented bug-free ROS code
- a README file, explaining how to run your code, and how to get it to achieve the desired behaviors.
- A ROS launch file, which should launch all the nodes of your project
- A media-friendly movie demonstrating your robot in action.

Total Desired Points from Basic Features: 70/70

Gathering and manipulating sound data from the Kinect microphone.

Desired points: 3/10

In order to gather data from the microphone, we realized early on that there wasn't a preinstalled package that was able to deal with the sound data. Openni is a package that is already installed with *ros* that is able to gather and manipulate data from the Kinect, but it only handles visual data. After some research, we found something called the HARK Project. On its documentation, the HARK Project is the system that aims to be an "auditory OpenCV." We believed this was perfect with what we were trying to achieve, but ran into multiple problems while installation. We followed many variations of installation instructions online while using the HARK documentation (http://www.hark.jp/wiki.cgi?page=HARK-KINECT) as the main guide, or, more specifically, the HARK installation guide made specifically for the turtlebot: (http://www.hark.jp/wiki.cgi?page=HARK-ROS-TURTLEBOT). While some of the installation guide complete the installation, there was no solid connection from the Kinect sensors and ros, and thus no data was able to be collected. Further research (http://answers.ros.org/question/48684/kinect-microphone-array-linux-hark-in-combination-with-openni/) suggests that there may be a problem with the compatibility of drivers. It also seems as though there are lots of compatibility issues between the different ros versions and HARK versions. The official documentation of the installation guide of HARK with Kinect is for Ubuntu 10.04 or later while saying

that installation with Ubuntu 12.04 is unstable, which suggests that an update hasn't happened in a while.

A* Path Finder

Desired points: 20/20

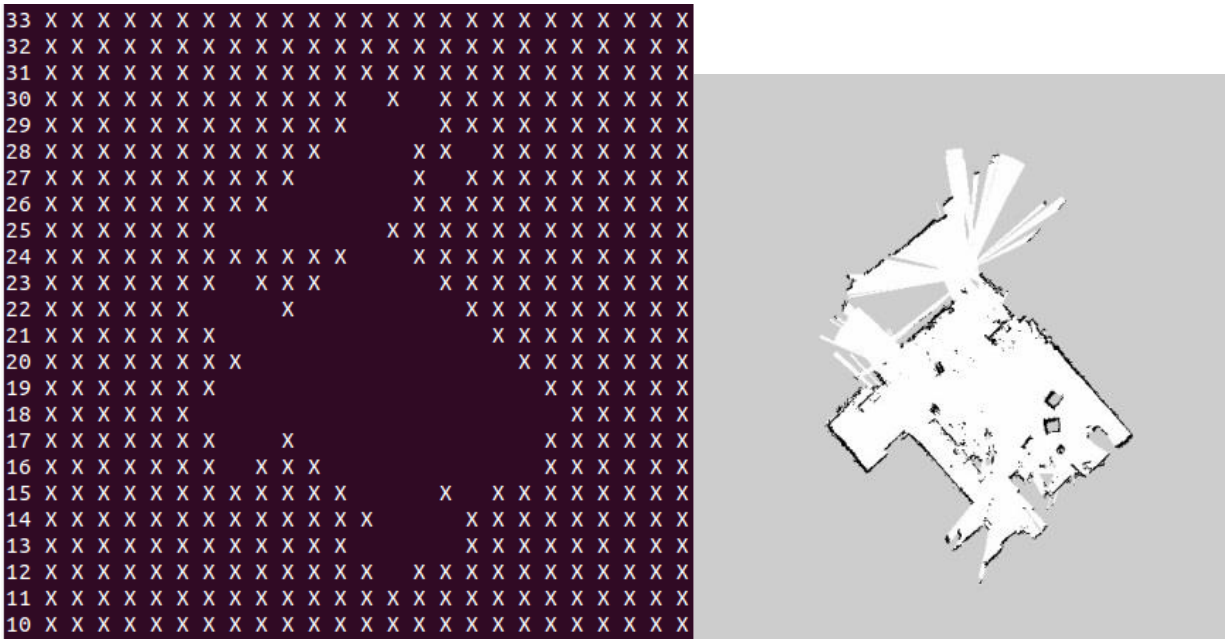
Our path finder first discretizes the map data into something we can manipulate. We load the map, and one of our nodes subscribes to the topic which holds the numeric data of the map. By subscribing to the /map topic, the data that we get is the numeric value of each pixel of the image in row-major order. There are three possible numerical values that comes from the .pgm map image: -1 for an unknown value, this value is for pixels that are beyond the actual walls of the image, 0 for a clear space, this value is for pixels that are mapped that doesn't have any obstacles, and 100 for an object, this value is for pixels that are mapped that has an obstacle.

Our next job was to create a new map with a lower resolution since the original map is quite large (700x500) in order to run our A* search algorithm at a reasonable time. We divided the map into 20x20 pixel grids. The metadata contained in the map image file says that the image is 0.05 meters / pixels. The value of the grid was determined by the majority function. If most of the pixels were clear space, the entire grid would be considered as clear space. Same goes for the other two possibilities. We created the new map in a 2d array. This is all done in GridConverter.py.

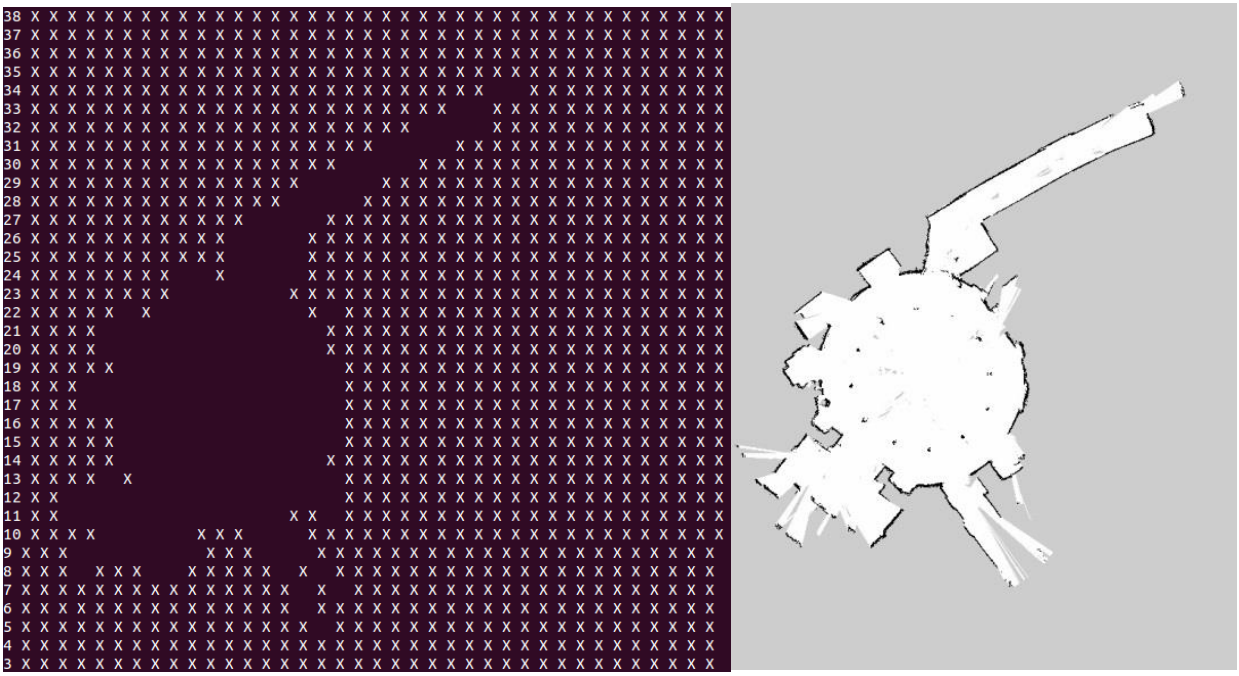
The GridConverter than uses an AStarRunner object in AstarPathFinder.py to find a path given the new map. Using John Rieffel's A* search algorithm code, we were able to pass in the new map, the starting location, and the target location, and get the path. With that path, we converted each step from the path produced into a series of Twist commands for the turtlebot using a PathToTwist object. We then publish the twist commands one-by-one while giving sufficient time to complete each command before the next one is published.

Maps printed in the terminal that helps determine coordinates:

CROCHET Lab:



Olin Rotunda:



Limitations of our A* Path Finder

Because we don't have a visualization controller like RVIZ, we have no other way of inputing the robot's starting coordinate and goal coordinate other than by hard coding it. When the node is run, the node prints out the map as a graph in ASCII format with which the user can then find its coordinates. Since the map is simplified (lower resolution), this makes it very difficult to pinpoint the exact location. Another limitation is that the series of commands we send to the robot doesn't account for odometry error. Given a path plan, the robot will follow the commands blindly without correcting for error. After a path is followed, it might be in a completely different coordinate than it thinks it is in. Since we don't have a way of dynamically updating the map while in motion, this was a major problem when trying to test the path finder, especially in longer distances. Finally, we couldn't find a way of incorporating the path finder into our FSM. Because a reaction from a bump event or an avoid event would divert the robot from its path, the robot would be completely lost after a change in a state. Again, because we don't have a way of dynamically updating the map, we decided not to incorporate the path finder into the FSM but rather as a separate function on its own.

Twitter integration

Desired points: 10/10

https://twitter.com/UnionTurtleBot

Tweets the robot's command that it is executing while running its navigation path.

Manipulation of turtlebot's LED lights

Desired points 5/5

LED lights on the turtlebot changes as state is changed in the FSM. (See Video)

Operation outside of CROCHET Lab

Desired points 5/5

With the map of the Olin Rotunda, the navigation takes it as input and is able to find a path given starting and goal coordinates. (See Video)

Total Desired Points from Additional Features: 43/50 points

Total Desired Points from Project: 113 points

Youtube link to videos:

A* path finding in Olin

https://www.youtube.com/watch?v=5ukKZq1tFP0

Move and Avoid in Olin

https://www.youtube.com/watch?v=23H4iTBfUss

Chase function in Olin with changing LED lights

https://www.youtube.com/watch?v=JC64ZOh3j3w