CSCI 3302 Final Project

Trevor Buck
Elly Landrum
Gerritt Luoma
Meriah Mackillip

Abstract:

In this final project we are beginning to tackle the issue of retrieving objects from separate rooms through the use of the Fetch robot. The Fetch robot, produced by Fetch Robotics, is a robot fit with wheels, a camera, laser sensors, and an arm. We will put our Fetch bot into an already mapped maze with 4 tables at the end of specific pathways. On these tables we will randomly place three colored objects, red green and blue, on three of the tables leaving one table without an object. The Fetch robot will then be given a color to search for and it will move to each of the four tables until it finds the specific object and touches it with its arm. Our goal is to accomplish this task through the use of basic color filtering and inverse odometry.

Introduction and Background:

In the world today, there is a large growth in online shopping. This increase leads to an increase in warehouses, and a greater demand for warehouse workers. However, these jobs are often dangerous and undesirable for people, so it is worthwhile to create a robotic platform that could itself do such work. A robot of this type would be told to go get an object. The robot would need to search the warehouse for the object and then retrieve it to the boxing/shipping area. In this way, we can save humans trouble and effort, in addition to perhaps saving money on labor.

This task has many components that we must complete. First, the robot has multiple stages it must complete. It must 1. Navigate around the map 2. Use its camera to try to detect objects 3. Interact with the object. 4. Return to base area. We will use a state machine to encompass these different stages. These different stages each have different required functionality. Fetch needs a map to navigate the warehouse, code for moving the base, identifying desired objects, and navigating the arm to pick up items. We will map the environment by manually driving the robot around. Once we have this map, we will add some "padding" to the wall so that the fetch robot cannot navigate too close to them. The base navigation is completed by using Dijkstra's to find the shortest (and thus approximating fastest) path to the possible object locations. The object identification is handled by a very simple form of computer vision. Since all objects and tables are in uniform locations we are able to use a general movement plan and only scan for the object when it will be in sight of the robot. The arm movement will be just touching the object once it has been identified and is close enough to it. MoveIt will be used to implement this.

Methods:

Arm Control:

This section was completed by Trevor Buck. The Fetch Robot has 7 degrees of freedom, has a length of 940.5mm when fully extended and can move at 1 m/s. Taking these specifications into account our robot will need to stop a previously specified distance away from the correctly colored object, reach out and touch it. To provide the correct trajectories for the arm to have its end effector touch the object we will be using MoveIt! MoveIt! will provide a path for the joints to follow so the arm can move from one position to another. It takes into account the collisions the arm will have with itself by labeling different pieces as adjacent links or never in collision, so we will not have to worry about the collisions the arm will have with itself. MoveIt! will also allow us to avoid obstacles within our environment such as the table the object is placed on, the path planning algorithm will take into account the robots surroundings and not plan a path that will lead to a collision. We believe this to be the best implementation of this because it is simple and will plan a path for us.

Base Navigation:

This section of the project was completed by Elly Landrum. The robot needs to move around the maze to allow the camera to identify objects and to get close enough to the objects so that the arm can interact with them. At the start, the robot knows the location of the tables and the maze. Thus, the robot just needs to navigate to the different tables until they find the table with the right object.

At the beginning of the project, we used Dijkstras to move between the tables. In order to do this search, the robot iterates over the location of the four tables. For each table location, the robot performs Dijkstras to determine the shortest path it can take to reach the table. The robot then navigates along this path until it has reached the table. Then, if the visual system does not tell the base navigation that the object is on that table, the robot picks the next table to go to. It uses Dijkstras to find the path to the next table, it navigates to the table, then it repeats.

After coding and testing Dijkstra's algorithm, we realized that it was very slow. First, it took a long time (about 30 seconds) to calculate the route it would need to take to reach its destination. Second, the route it plotted was often zig-zagged; this route took a long time to traverse as the robot had to turn every time to get to a new waypoint. It would work, but it would take a while. It was discovered that it was much faster to handpick some custom waypoints for the robot to go to. With the map given, we could go from (0, 0) to every table with only two waypoints, which means the robot only has to only turn once. This allowed for a faster completion of our goal.

Mapping:

The mapping was implemented by Meriah Mackillip. The map that we have generated for this project is an 8x8 meter maze composed of walls and four tables. The Fetch bot will begin its operation already having an array representation of the map generated for an ease of traversal throughout the maze. Every path and wall that was used takes up the same amount of floor space to increase the accuracy of the array representation of the map. Starting off we had each cell be two squares wide. As we continued working we realized that it would be easier to avoid wall collision if we inflated each cell and created padding. The easiest way we thought to do this was to have each cell that was adjacent to every wall consider itself a wall so we were easily able to increase the wall size.

The other use for standardizing the amount of space taken by the components of our maze was to provide waypoints in the direct center of the traversable path to reduce the complexity of our object detection system.

Object Detection:

Our object detection was implemented by Gerritt Luoma. We decided to utilize a very basic implementation of computer vision to accomplish our task. The only time our computer vision will actively filter out desired colors within the image provided from our camera is during stage 1. Stage one begins when our movement controller has moved our bot to the same row or column as our target (depending on direction of approach) and begins moving towards it. This ensures that both the table and target (if on the table) will be in sight of the camera and available for detection. If the camera recognizes the object, it will switch our robot's state to stage 2 where it will move to around half a meter away from the table and tap the object with its arm before moving back to the starting position. If our robot makes it to within half a meter of the table without sensing the target, a signal will be sent to the movement controller to return to stage 0 and move towards the next table. Our method of determining if the target is on the table is by applying a simple filter to the image filtering out only the desired target's color. If there's one or more blobs in the generated mask after filtering our image, the target is present and the bot should approach and tap it. We have been able to use this efficient, low strain implementation due to our controlled, mapped environment with a constant background completely different from the desired colors of our targets.

Results and Discussion:

Overall, our project turned out fairly well. The mapping, object detection, and base movement all worked well. The robot successfully completed the task we wished it to accomplish. However, there were still some problems.

Using Dijkstra's for base movement worked perfectly but was incredibly slow. To solve this, we found a method that was much quicker. This temporary solution was to hand pick specific waypoints generated by our Dijkstra's algorithm. One downside of picking waypoints by hand, however, is that the code would only work on one map. It is not generalized to a variety of locations. A future attempt at this problem might attempt to find a path that prioritized not the shortest distance, but instead least number of turns since turning caused the robot to slow down the most. An algorithm that might work well for this problem is the visibility graph method. This method goes in straight lines between the corners of obstacles. With the padding in the map, we could perform this path planning without worrying about hitting the obstacles. Furthermore, this method generates many straight lines for the path, and would probably run the quickest.

One potential issue for our object identification is the fact that the real world will not be uniform during all trials. Because of this, the robot wouldn’t use a generalized movement and object recognition plan but would require more advanced object recognition paired with laser scans to provide distance measurements for the arm. Similarly, most areas that this object would operate in won’t have walls and floors consisting of the same color. Since we are using basic color detection, our robot could be prone to false positives originating from the background. For our proof of concept, however, our current solution is adequate for the job.

.

Link to Github Repo:

https://github.com/LEL-15/RoboRally