COMS 6731 Humanoid Robots
Final Report
May 2017

Fetch Waiter: Delivery Time Optimization

Team 4
Daniel Hong (sh3266)
Antong Liu (al3606)
Calvin Li (ctl2124)

Abstract

In class, students participated in a demo of a Fetch robot performing a grasping task on a stationary object, but one problem that kept occurring was, given the position of the robot relative to the object, there was no reachable grasp. We work around this possibility by having the robot locate an object on a table/counter, and then strategically position itself so that the object is reachable. This capability would be very useful for a household humanoid robot, in particular one that would work in a kitchen and might have to locate and grasp objects all around a table or counter. The robot should also deliver the object to some goal location around the edge of the table (e.g. to a customer). Furthermore, the robot should be able to navigate a cluttered environment to get to the object, and then navigate through the environment once more to deliver it to another table. We optimize the total amount of time it takes to assume a suitable position to grasp the object, to grasp the object, and then to deliver the object. At first, we approximate this only by minimizing the distance traveled to the suitable position in addition to the distance traveled to the goal afterward. Then, we also take into account for the differing amounts of time that any given grasp itself may take by precomputing some simulations of the various possible grasps the robot will encounter. We also turn the robot into a "waiter" which finds the shortest path among obstacles, such as tables in a restaurant, in order to pick-up and then finally deliver an object somewhere else.

Introduction

This report addresses optimal path finding and object delivery problems for a humanoid robot that has to find and collect an object of a predefined shape and move it to a desired position in minimum time. We use Fetch, a mobile humanoid robot with a 7 degrees of freedom arm that comes equipped with a variety of sensors that empower it to work well in home and lab environments, to accomplish our tasks. Leveraging grasp capabilities and head camera sensors of the Fetch robot, Team 4 presents **Fetch Waiter**. Fetch Waiter is a delivery time optimization system that automatically estimates the least-time-consuming path to deliver a predefined object from one location to another. Deliverables of this path optimization project extends beyond object detection and grasping; our path planning and delivery pipeline caters to Fetch by forming a set of collision-free candidate paths and picking the best route by optimizing distance through Dijkstra's algorithm as well as grasp time through precomputation.

We demonstrate Fetch Waiter in a real-life application through a simulation environment of a restaurant. Traditional graph search and path finding algorithms account for just distance as the cost to optimize; although we also use such a path-finding algorithm, such an approach from the human-movement perspective overlooks the considerable amounts of time spent during the grasp planning and execution stage. We propose that optimizing an objective function over grasp time as well as distance is more effective and realistic for a humanoid robot rather than calculating just the shortest collision-free path. Although the simulation was targeted for a restaurant environment, the same system is applicable and appropriate for assisting immobile patients or the elderly who require timely responses.

We add our contributions to existing functionalities from Fetch's demo code. Existing code demonstrates how Fetch is flexible enough to map and navigate its environment with SLAM and also plan grasps with inverse kinematics. While the features are extensive, we sought to augment the Fetch robot demo in three ways.

Approach 1:

We first present a vision-driven solution for automatic object detection for a specific situation where the customer sitting at a large table wishes to grab an object on that same table without informing the robot of the exact location of the object. Based on position space discretization around the table, Fetch uses his vision system to locate, grasp, and deliver the object to the customer in minimal time. The vision-driven solution enables the Fetch robot to autonomously navigate within a generalized bounding area (a single table) in search of its target object without human teleoperation.

Approach 2:

Due to the high computational cost and completion time of this solution, we alternatively propose a precomputed-time-driven approach based on robot position adjustment. Given the coordinates of the object, we use the precomputed list of grasp times at numerous discretized robot positions around the object to navigate the robot to the optimal grasp location. This precomputed-time-driven solution empirically estimates efficient and reliable grasps that can be performed by the robot to minimize time lost on unsuccessful or inefficient grasps to offer task completion in shorter time. The robot then delivers the object to another location on the table, just as in Approach 1.

Approach 3:

The first two approaches are constrained to simply picking up an object and delivering it to another location on the same, possibly very large, table. In order to account for a "restaurant" type setting, the third augmentation to the demo code picks up an object at one table and delivering it to another table. It involves constructing a visibility graph of obstacles in the environment (in our case, rectangles representing tables), and performing Dijkstra's algorithm to determine the shortest path, using Euclidean distance between points as edge weights [4].

Objective Function

Our path planning algorithm optimizes the following objective function, taking into consideration both the distance and time:

Distance from robot to *s* + distance from *s* to goal + time it takes for a grasp at *s*

The variable ***s*** in the function is a specified location around the object where the robot should be to perform the grasp. This location is calculated by searching through our precomputed grasp time vectors. We optimize the function by reducing each term independently. The shortest path from robot's initial position to ***s*** and the shortest path from s to the customer/delivery location are found independently.

Vision-Driven Solution and Delivery Around Single Table (Approach 1)

In our project, we used a blue cube as our target object. In order for the Fetch robot to autonomously identify a target object, we used the vision system from Fetch's head camera to survey the scene and segment the resultant point cloud. Due to the Fetch robot's close integration with the ROS operating system, it was possible to perform this segmentation using existing libraries. The result of such a segmentation is a list of clustered points, with each cluster belonging to the same physical object. The Fetch robot then decides which clusters of points represented objects that were of interest. In order to detect the location of the cube on the table, we received data from Fetch's nodes. From "head_camera" node, we are able to subscribe to obtain point cloud data objects from "FindGraspableObjects" contained in "grasping_msgs.msg" rosmsg. This allows us to identify the target cube. To demonstrate automatic object detection, Fetch moves around discretized locations around the table and checks if the cube is present in a while loop. Once the cube is found, our baseline Waiter algorithm computes the shortest path from the robot's location ***s*** to the customer. We simulate this elementary solution in a simple environment to gauge the success of object detection and our initial object delivery method. The path finding algorithm used in this vision-driven solution grows the table and finds the shortest collision-free path around a single table. The single-table path finding algorithm in this approach is the same as the one in Approach 2, where it is described in more detail.

Precomputed-Time-Driven Solution and Delivery Around Single Table (Approach 2)

We ran the simulation repeatedly to sample the average grasping times for various positions of the robot and object. We saved these grasping times as values corresponding to the vector between the base of the robot and the position of the cube. During the experiment, we used these precomputed times to decide the ideal position to attempt the grasp from. This approach captured some of the variability in grasp success rates since a failed grasp added significantly more time to the whole process. This penalty increased the cost associated with grasping from that position.

After the robot moved into place, the Fetch robot planned a grasp and moved its arm into position to execute the grasp. This was accomplished using the MoveIt! ROS library and GraspIt planner. [1] Once the object was attached to the robot via a grasp, the robot moved to the goal

position by using the MoveIt! ROS library. To avoid some cases where the default MoveIt trajectory would become stuck or route inefficiently, we decomposed the path between the robot and its destination. This decomposed path consisted of one to three intermediate points, which guided the robot to its final destination along a direct route. Our method was to use some basic assumptions about the layout of the environment to calculate in constant-time the path. The geometry of the setup allows us to determine the shortest path by comparing the coordinates of the robot and customer with the width and length of the table and determining which corners of the table to navigate around in order to most quickly move to another location around the table. The corners of the table are "grown" out from the original table to account for the width of the robot.

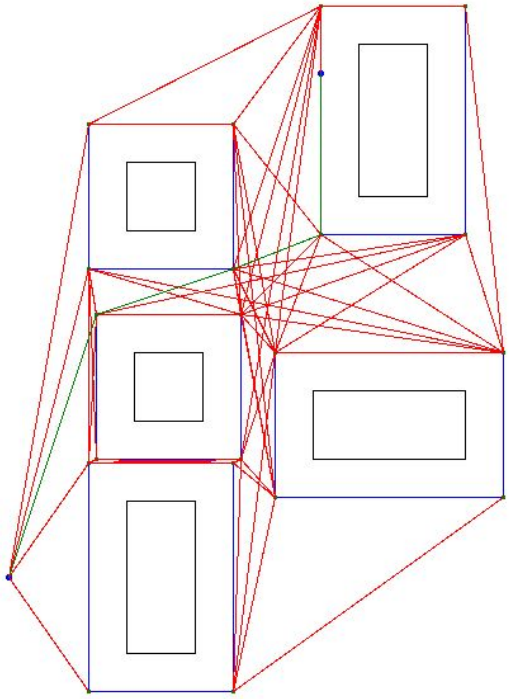

Figure 1: Visibility graph on grown obstacles shown, with Dijkstra's shortest path in green.

Precomputed-Time-Driven Solution and Delivery Among Several Tables (Approach 3)

This approach involves constructing a visibility graph of the obstacles, and growing the obstacles by the width of the robot to ensure the robot does not collide with any of them. The corners of the grown obstacles are used as vertices of the graph, as well as the robot start location and the robot goal location. Dijkstra's algorithm is then run on the graph, using Euclidean distance between vertices as edge weights, from the start to goal in order to find the shortest collision-free path, as seen in Figure 1. Dijkstra's algorithm is run twice in our implementation. First, the goal location is determined by using our precomputed grasp vector file to find the best

location to perform a grasp on the object around the table that contains the object (same as in approach 2), and then Dijkstra's algorithm is run from the robot start point to this pick-up location. Then, once the object has been grasped, the goal point becomes whatever predetermined delivery destination the user wants, and Dijkstra's is run once again on the visibility graph with the new start and goal points (from pick-up location to delivery location).

Challenges

System/infrastructure: We initially met a lot of challenges in setting up and getting used to the ROS environment. We also ran into various CLIC machine issues after having struggled to set things up on our own machines, such as ROS logs building up so much that the CLIC disk quota would be exceeded. The Gazebo simulation failed to run remotely through CLIC, as well as on an Ubuntu VM, so we eventually settled on working physically in the CLIC lab.

Grasp planning: Our method of precomputing grasps only took into account the relative position of the robot to the object. However, the grasp planning provided by moveit is nondeterministic, and there are many other factors that go into a planned grasp that we could not account for. Therefore, our method still experiences failed grasps on around 50% of runs. According to the Fetch Gazebo documentation, the simulated grasp planning may perform even less ideally than in real-life due to a less well-tuned arm and fingers [3]. Ideally, in the future we would be able to have more control over grasp planning and precompute more aspects of the ideal grasp given a predetermined object in order to better ensure a fast, successful grasp. Also, we would have loved to test our code on a real-life robot and environment, but could not due to time constraints.

Path planning: Our use of a visibility graph on grown obstacles in Approach 3 helped Fetch navigate the environment very smoothly. However, the moveit library also has some strange behaviors when navigating the robot, as it often determines that the robot is stuck even when it clearly is not in collision with any obstacles. Therefore, we decided to grow the obstacles conservatively by growing them by a little bit more than the robot width, as well as growing around the obstacles uniformly. This helped ensure the robot takes a smoother path that involves less of the collision recovery behavior that moveit often encounters when navigating Fetch around the environment. Even so, the nondeterministic nature of navigating with moveit makes it so that randomly on certain runs the robot will take a trajectory from one point to another that is not ideal, which then causes the robot to take a little extra time to recover before moving on to the next point in the planned path.

Proposal modification: Our initial proposal involved tracking and grasping of a moving object. After initial research, we devised an interesting hypothesis that if the moving object and the trailing humanoid robot are moving at the same speed, the relative positions of the object and the robot must remain the same. Therefore, the robot should be able to perform grasp planning and execution in motion successfully. Upon discussions with the TAs, we diverted from this proposal and resubmitted a revised version with our current objectives for several reasons. The TAs raised concerns that the tracking and grasping task may be very difficult and unrealistic. Firstly, Fetch's hand is restricted to only pick up small objects. Detecting and tracking a small object moving on the ground in a cluttered environment would be difficult, and even impossible with Fetch's

camera/sensor specifications. Secondly, there are limits due to Fetch's specifications including arm length, speed, and processing capabilities of conducting object tracking and grasping concurrently.

Results

Both vision and time based methods introduced in this experiment lead to successful pick and deliver pipelines. The first attached video shows the Fetch robot moving to a location calculated by our relative vector approach. As mentioned in our challenges, due to fortuitous uncontrollable Fetch sensor performance, the block slips from the robot's grasp just before the placing of the object is completed, but the video demonstrates an otherwise successful completion of the entire pick and deliver task. In the second video the Fetch robot moves directly to the optimal location from which our precomputed vectors predict a rapid grasp. From there it completes the rest of the pick and deliver task similarly to before. Finally, in the third video, Fetch navigates through a cluttered environment of tables in order to reach a pick-up location determined by our precomputed grasp vectors, grasps the object, and then finally navigates through the environment once again to another table where to places the object.

Conclusion

Three approaches to accomplish humanoid delivery optimization have been addressed. We demonstrated that our three contributions were reasonable ways of improving the Fetch robot's ability to autonomously complete a pick-and-deliver task while minimizing the time spent. Our group divided the responsibilities as follows:

- Daniel Hong: pipeline implementation, research
- Antong Liu: algorithm development, research
- Calvin Li: ROS integration, algorithm development, testing

References

[1] Planners available in moveit!
[2] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. Fetch & freight: Standard platforms for service robot applications. Fetch Robotics Inc., 2016.
[3] http://docs.fetchrobotics.com/gazebo.html
[4] http://www1.cs.columbia.edu/~allen/F16/NOTES/lozanogrown.pdf
[5] http://ieeexplore.ieee.org/document/7402118/