Motion Planning under Sensing Uncertainty

MECH 598 Project

Carlos Quintero Peña

05/01/2020

OBJECTIVE

The main idea of this project is to create a realistic simulation of a Fetch robot that solves the task of planning and executing a trajectory to grasp an object on top of a table when the scene representation comes from on-board short-range sensors and therefore it may be incomplete.

WHY MOTION PLANNING?

During the course we learned how to design and analyze different types of robot manipulators, starting from their kinematics all the way up to the control. However, we usually either assumed that we had access to a trajectory generator or we would generate trajectories by simple interpolation in task space. Although this approach proved practical for the problems at hand, it leaves out certain aspects for some applications that are required to achieve a realistic simulation of a robot manipulator that interacts in a given scene. Generally speaking, it is not practical having a human generating trajectories for the robot to execute, for example for general-purpose service robots. Also, having to solve the inverse kinematics for task-space pose may be too expensive, especially for fine-grained trajectories. Finally, we need a way to handle collision avoidance. Even if the obstacles are static and known beforehand, there is a need to avoid joint configurations that make the robot's links collide among them.

The problem of Motion Planning has been extensively studied during the last decades. In a nutshell, the idea is the following: given an initial and goal robot joint configurations (the former can be the current robot joint positions measured by sensors in the actuators and the latter can be computed by solving an inverse kinematic (IK) problem only for the final end-effector pose), compute a path in **joint space** that connects them and that avoids collisions in task space. One successful approach to solve the problem has been Sampling-based Motion Planners (SBMP). These methods solve the problem by randomly sampling the joint space in order to capture its connectivity by creating roadmaps that can be used to find feasible paths. Since the shape of obstacles in the joint space is not known and can not be efficiently found, SBMP methods rely on collision checking routines that for every joint configuration perform forward kinematics (FK) and check whether any point in the robot's body collides with any other point in the scene (including other parts of the robot).

The problem of motion planning is a natural application in the course topics since it heavily uses all the concepts learned during the class and it allows us to simulate situations that are closer to a real-life application. In particular, IK is used to find goal joint configurations and FK is used for every sample to check for collisions.

TECHNICAL DESCRIPTION OF THE PROJECT

The objective of the project is to simulate the Fetch robot in a realistic environment considering uncertainty. In order to accomplish this we needed to take into account the complexity of a real robotic platform, starting with the topics covered during the course, i.e., kinematics, dynamics, control, but also others that were either briefly discussed or not at all, such as sensing, localization, motion planning, etc. Instead of

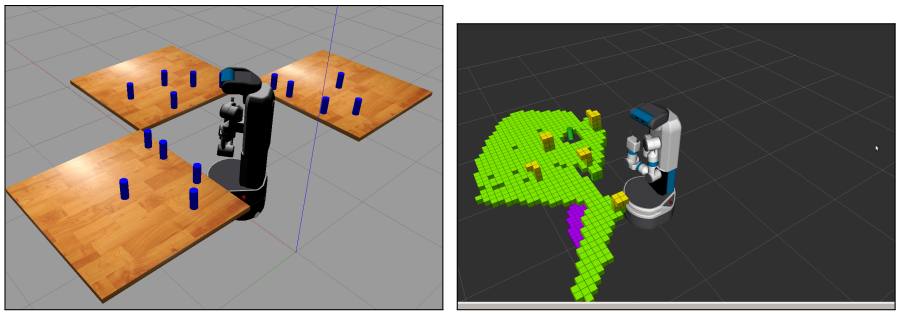
building or analyzing all of these from scratch for the Fetch robot, we used a large software stack based on ROS (Robot Operating System). In the project, all of these were integrated in order to solve the problem of successfully planning and executing a grasping task in the presence of uncertain obstacles. The following are the main software components of the solution and a brief description of what was done in this project with each one:

- **Gazebo:** It is the physics simulator where the world and robot are modeled. For the project I created the testing scenes by creating first objects (cylinders in this case) that represent cans and wooden tables. Each graspable object required a grasping pose that the robot knows beforehand that is used to compute the goal configuration for every can. The poses in ROS are represented as a $3D$ vector that represents the position and a $4D$ vector that represents its orientation as a quaternion. Figure 1a shows an example of one Gazebo scene
- **tf2**: It is a ROS package that provided all the transformations between the different frames in the world. The robot geometry is already known for the the package and the transformations are given as abstract objects that can be operated as we operate transformation matrices. We used these transformations to compute the frame origins of the robot's links that were part of the planned trajectory
- **MoveIt:** It is a ROS package that provides access to a set of sampling-based planners available in OMPL. Plans are represented as a sequence of robot states (joint configurations) that are created from the group of joints that is used to plan. For this project we created a group for the 7 arm joints and the torso and therefore, each state could be represented as an 8-dimensional vector. This package also provides controller functions for executing the planned trajectories. The trajectory was required to be time-parameterized before passing it to the controller. The controller also provides a way to receive feedback on whether the given trajectory was successfully executed or not. This was required for the experiments
- **Octomap:** It is a library that efficiently stores and manages data that comes from cloud points such as those captured with a RGB-D camera. The octomap represents the world by classifying parts of the space as free, mixed (free/occupied) or unknown. It creates boxes and each box is labeled as one of the possibilities. If the box is empty it stores it with a large resolution, if the box is mixed, it further split it into smaller boxes and it keeps doing that until all are free or occupied or a resolution limit is reached. It was used in this project to represent the scenes. In the experiments, we used a query function to know whether a given point in space was already explored or unknown. Figure 1b shows a scene representation using octomaps.
- **Quickhull:** It is a library that provides fast computation of the convex hull of a set of points in three dimensions. It was required in the project to estimate the swept-out volume of a given trajectory

THE EXPERIMENTS

In order to succeed in the task of planning and executing a motion to grasp an object with sensor data, we needed to improve the scene representation. In many cases, it turned out that the robot was capable of successfully finding a plan but it failed in executing it because the scene was incomplete. This happens because the planner uses unknown space as free. The trivial solution of changing all the unknown space to occupied would not work because it becomes extremely likely that no path would be found since only a small part of the space is actually known by the robot. This assumption is also necessary to allow robots that do not know their entire environment to be able to explore it.

Figure 2 shows a diagram of the general proposed solution. The idea is to start a plan with the current known scene and then use this plan to extract points that the robot needs to explore before executing the trajectory to make sure that it will succeed (it won't collide). After that, this points are examined to determine whether they are known or not and we command the robot to look at the set of those points that are unknown. Finally, the robot will create a new plan with the updated scene and will repeat this



(a) (b)

Figure 1: a) Scene and Fetch robot in Gazebo simulator. b) Octomap representation of the scene obtained by the on-board RGB-D camera of the robot

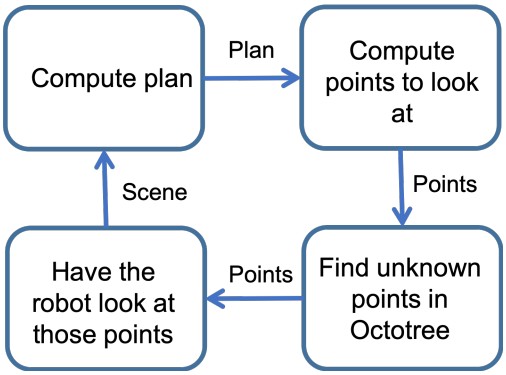


Figure 2: Diagram showing the main steps of the proposed methodology

process until finding a safe plan (one where all the interesting points are in known space).

In order to decide which points are interesting to look at, we have explored four different strategies, described below:

- **Object**: Instead of using the planned trajectory, the robot only looks at the object
- **Fixed points:** The robot looks at two points at each side of the object
- **Frame origins:** The robot considers all the arm's link origins for all joint configurations of the planned path
- **Swept-out volume:** The robot considers the volume swept-out by the arm if the trajectory were executed. The volume is estimated by computing the convex hull of consecutive joint configurations in the planned path

Figure 3 shows an image of the robot simulation looking at unknown points of the swept-out volume marked as red arrows. In order to compare the four strategies, we designed the following experiment: we place the robot base in a random position and orientation close to the tables. We randomly choose one can from all in the scene. Then, the robot implements the proposed methodology to create a plan to grasp the selected object using one of the strategies described above and finally the robot executes the trajectory. At that point, there are several possible outputs; the robot is able to succeed in the execution of the trajectory (i.e., it does not collide with obstacles and it reaches the grasping pose), the robot fails in executing the trajectory, the robot can not find a path to solve the problem at hand. For the cases of **frame origins** and **swept-out volume** strategies, the robot only executes the trajectory when the planner outputs a plan and

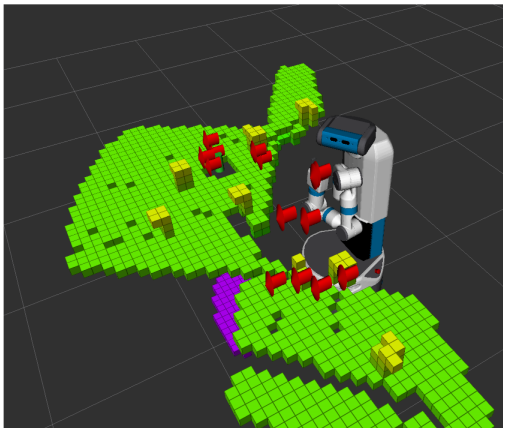


Figure 3: Fetch robot looking at interesting unknown points of the swept-out volume of the computed trajectory

Table 1: Results of 50 runs for the proposed methodology for completing the scene for the 4 different explored strategies

	Object	Points	Origins	Convex
Time	26 ± 10.9	23.6 ± 10.44	71.9 ± 44	62.8 ± 39.2
Success	22	25	42	41
Fails	28	25	8	9
No path	0	0	3	7

all the interesting points are in the known region of the space. If the robot makes three planning attempts and it is still not able to find a completely safe path, it takes the risk and executes the current, potentially unsafe plan. This may or may not end up in collision but it is done to make sure that the algorithm won't get stuck in infinite loops and also to avoid excessively large simulation times, especially because there might be points in space that could be occluded.

Each strategy was tested when 50 trajectories were executed and the time taken to complete the scene was measured for every strategy. Each motion planning problem is independent of the previous, meaning that the knowledge of the previous scene is not used before solving the next problem.

RESULTS

Table 1 show the results of the 50 runs for each strategy. The time row shows the mean time and standard deviation in seconds. As expected, the simple **object** and **fixed points** strategies take much less time in looking before executing the planned trajectory since they choose fixed points. However, it can also be seen that their probability of collision is much higher; much more than half of the different trials were not successfully solved. The **frame origins** and **swept-out volume** strategies effectively increase the probability of succeeding in the task execution since they take into account important points in the planned trajectory. Also, even though they take much more time than the simple strategies, this time is worth in applications where safety is a priority. Finally, these strategies can also create unsafe trajectories since a maximum number of attempts is considered. This number could be increased to further reduce the chances of obtaining unsafe trajectories at the expense of more time taken to look.

A video with several runs showing how the swept-out volume works can be found in https://youtu.be/mFQVLOuSuAE.

CONCLUSIONS

In this project we have implemented and integrated some of the most important concepts of robotics into a realistic simulation of an 8-DOF manipulator that senses its environment using short range sensors. To do this, we have used concepts from the class such as kinematics, dynamics and control and others such as motion planning, sensing and uncertainty. We have explored 4 different strategies to obtain missing information from incomplete scenes when the robot uses octomaps created from on-board sensors. The most successful ones use information from the trajectory's waypoints such as link's origins and swept-out volume. Finally, this project is an important step towards implementation utilities with the Fetch at our lab where we can successfully solve motion planning problems using only on-board sensors without previous information about the scene. Also, the future work of this project is to further investigate other ways to solve the problem by implementing more active-sensing strategies such as moving the torso or moving objects around if required to collect more information about the scene.

CODE

All the code developed for the project can be found in the folder attached. It is part of a ROS package called `robowflex_datasets_fetch` and it contains the following folders and files:

- `include`: contains header files of the project
- `launch`: contains the launch file to run the executable with all its parameters
- `parameters`: contains the objects, grasping poses and scenes for the simulator and its interface with ROS
- `scripts`: contains the `main` function of the experiments
- `src`: contains the source files of the Fetch library interface, including code developed in this project to compute the swept-out volume of the trajectory and the interesting points for the robot to look

The external dependencies can be found in the `CMakeLists.txt` file. All of them are open source except for Robowflex, which is a development from Kavraki's lab that provides additional functionalities from the OMPL planners that **MoveIt** does not.