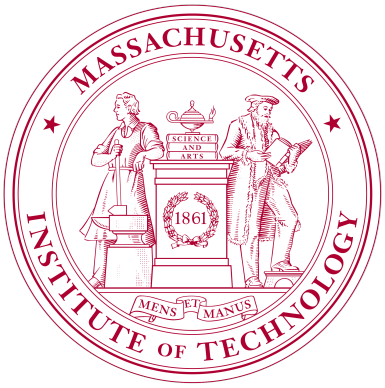


# PADM Fall 2023 Graduate Project

**Project Team:**

Michael Tibbs

Lorenzo Shaikewitz

Reinaldo Figueroa

**Course: 16.412**

**December 11, 2023**

# Contents

# 1. Introduction

This report will review all three parts of our Final Project for the Principles of Autonomy and Decision-Making class. These three parts are Activity Planning, Motion Planning, and Trajectory Optimization. For each section, we will review their implementation, main design decisions, challenges, and results. Then, we will establish meaningful comparisons between the methods we used to solve these problems.

# 2. Activity Planning

We are given a kitchen environment and a robot manipulator arm with a list of tasks we need to perform. These are:

1. Remove the sugar box from the top of the burner and place it on a nearby countertop.
2. Stow the spam box inside the red drawer below it.

For this task, we designed a PDDL domain and problem based on [5], and then implemented breadth-first search and enforced hill-climbing to search and find an optimal solution based on the definition of our PDDL problem. More specifically, our PDDL problem defines our initial and goal states, which is the goal we want to achieve in our activity planning task: sugar on the countertop and spam box in the drawer.

## 2.1 Implementation

While we discuss the specifics of our implementations on our project's README [2], we will mention some important details about our implementation in this section. Firstly, we established our PDDL domain by creating multiple types and their respective objects that we would need to define our problem:

1. surface(type): stove, countertop
2. openable(type): drawer
3. box(type): sugar, spam

Additionally, we defined a list of predicates that would help us identify concrete states in our problem, such as: *boxOn (box, surface)*, or *surfClear (surface)*. We then defined actions and the problem that needs to be solved using our PDDL elements. Next, we used search methods such as Breadth-first search (BFS) and Hill Climbing with a fast-forward heuristic inspired by [5] to solve the PDDL problem.

## 2.2 Assumptions and Design Decisions

To simplify the state space, we assumed there were only two surface positions (stove and countertop) as we would not need any others for the solver to find a solution. In the same spirit, while the kitchen could have many openable sections, we only defined the drawer where the spam box would need to go as our only openable object.

We used the fast-forward heuristic for our Hill Climbing search algorithm, as studied in class [5], in which we ignore the delete effects when transitioning from one state to the next using BFS. Hill Climbing + Fast-forward heuristic proved to be highly efficient at finding solutions quickly. See the README for more information [2].

### 2.3 Solver

For our solvers, we used an already built-in implementation of Breadth-First Search to test our kitchen environment and problem definition. We then implemented the Hill Climbing search with a fast-forward search heuristic, which performed exceptionally well on our defined PDDL problems.

### 2.4 Challenges

We found a few challenges when defining our predicates. More specifically, we had to go through a couple of iterations to find ways to check a predicate for all objects defined in our problem since we found that some of the PDDL built-in methods were not compatible with *pddl-parser*. For example, we had to re-define how to establish when an object could be held by the arm, knowing no other object was being held at that point. To solve this issue, we created a predicate, *armClear*, that would serve this purpose instead of iteratively checking for all objects if the arm was holding it.

### 2.5 Results

Our activity planning implementation can take the kitchen environment and a PDDL problem and use one of our search algorithms to find a solution if it exists. Additionally, we created a more complex problem to test that our PDDL kitchen environment was well-defined and that the search algorithms could solve more complicated scenarios (see [2]). This more challenging problem also allowed us to test the fast-forward heuristic we used in Hill climbing. Additionally, we validated our optimal activity plan by hand.

# 3. Motion Planning

This section discusses motion planning with the Rapidly Exploring Random Trees (RRT) algorithm, its integration with activity planning, assumptions, solver modifications, challenges, and results.

### 3.1 Implementation and Integration

RRT-based motion planning is integrated with the PDDL-based activity plan. It involves coordinating the arm's movements with the steps of the activity plan in a kitchen environment from the start (1) to the goal 2, focusing on collision avoidance, and adhering to joint limits as detailed in [2].

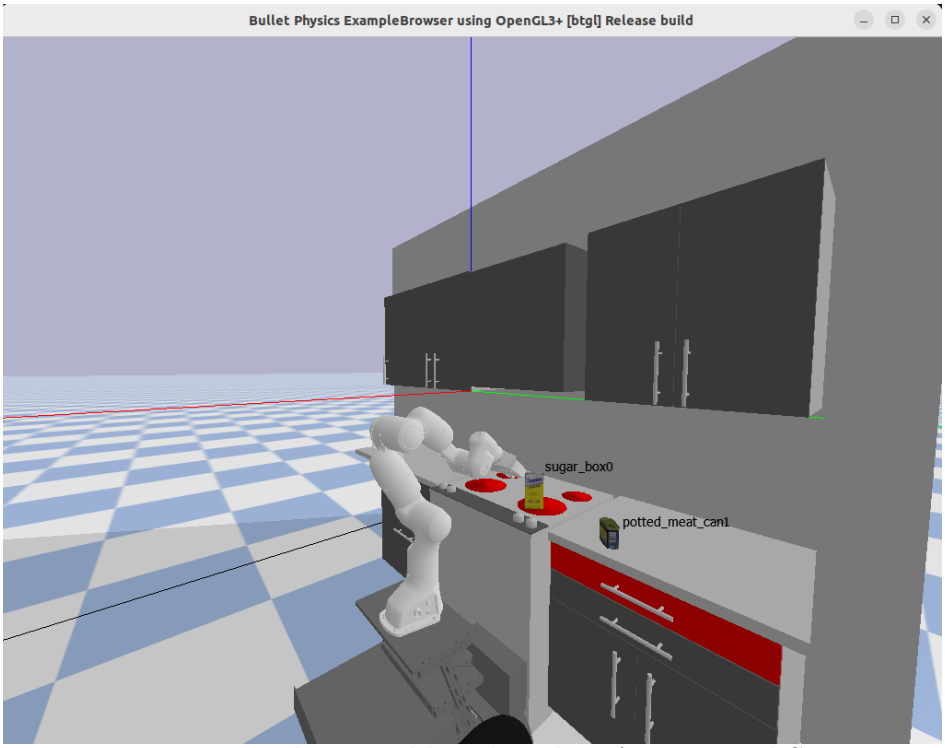

Figure 1: Kitchen World with Robot Arm: Initial State

## 3.2 Assumptions

Our motion planning with RRT in Pybullet relied on these assumptions: collision checks were limited to the robot arm and kitchen environment, not between the arm and objects like sugar and spam boxes. We assumed the goal region was close enough to pick up objects. Further, we simplified our model by not including gripping or drawer operations dynamics. Collision checks occurred only at specific joint angles, and we assumed a collision-free path if no issues were detected at these points. Additionally, the robot's initial position is assumed to be next to the kitchen counter. Finally, we assumed robot movements as a part of collision checking are not considered, only the final RRT trajectory. Please see the README for more details [2].

## 3.3 Design Decisions

Design decisions included using RRT due to learning it in class [3], having existing code from the PSET, and its ease of implementation with high-dimensional robots. A goal-region tolerance of 1.0° and a 20% goal bias were chosen to ensure the proximity of the gripper to the object and for quicker convergence in a low-density obstacle environment. Collision checking was performed between joint angles at 0.5° increments, ensuring no collisions between points and improving computational efficiency. RRT's max step set between $x_{nearest}$ and $x_{new}$ was set to 5° to ensure minimal collisions, reduce jerky motions, and converge faster. Finally, maximum iterations were set to 10,000 even though trajectories were solved in about 300 iterations.

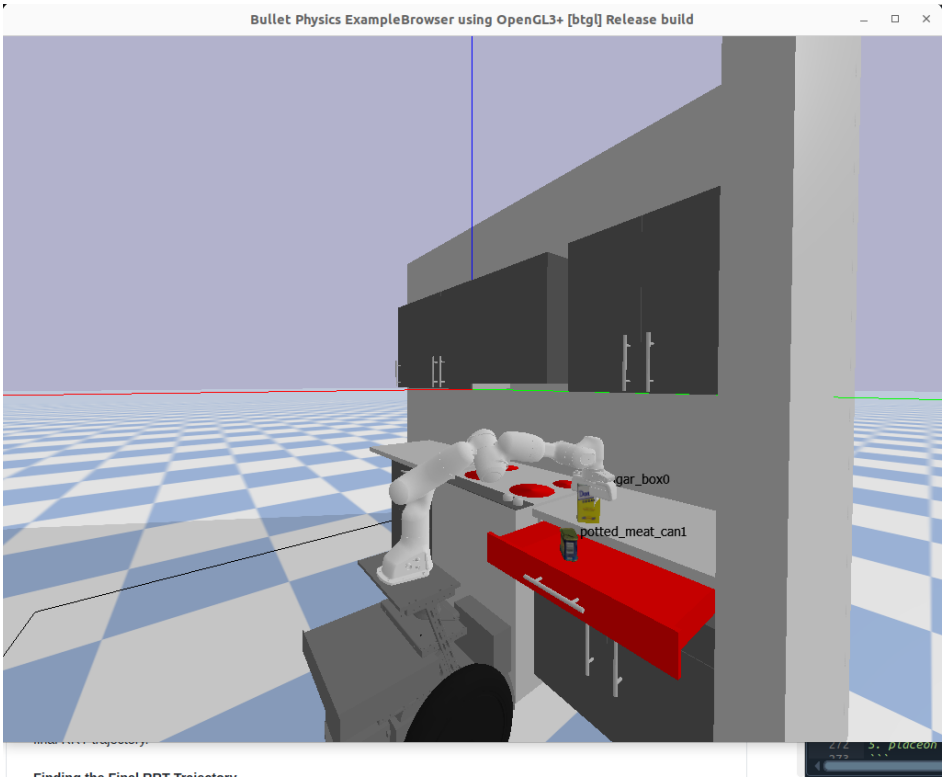

Figure 2: Kitchen World with Robot Arm: Final State

## 3.4 Solver

The RRT algorithm, adapted from PSET 5 2-DOF code to 7-DOF for the robot arm, involves querying the kitchen environment for joint angles and mapping PDDL objects to these angles. While calculating random points ($x_{rand}$) and finding nearest ($x_{nearest}$) and new ($x_{new}$) points is similar to the original 2-DOF version, it includes joint angle limits and collision checking. The path simulation integrates PDDL, matches gripper and object positions, and opens the drawer with a hard-coded trajectory, ensuring alignment with the robot arm's motion. More details are in the README [2]. A hard-coded drawer opening trajectory was chosen because the drawer opens in one direction, so the RRT jerky motion would not realistically work for this.

## 3.5 Challenges

Integrating the PDDL with our motion planning was a significant challenge, particularly in mapping PDDL objects to appropriate RRT start and goal joint angles. Although tuning RRT parameters wasn't overly complicated, the integration process required experimentation and adjustment. Another challenge was developing a method to open the drawer smoothly, requiring alignment of the drawer's motion with the robot arm's joint angles. Experimentation was also needed to realistically simulate the gripping and placement of objects, aiming for a smooth visual representation in the simulation. Surprisingly, RRT constraints were solved quickly to ensure collision avoidance and stay within joint angle limits. In general, the RRT algorithm was less challenging than working within PDDL and Pybullet.

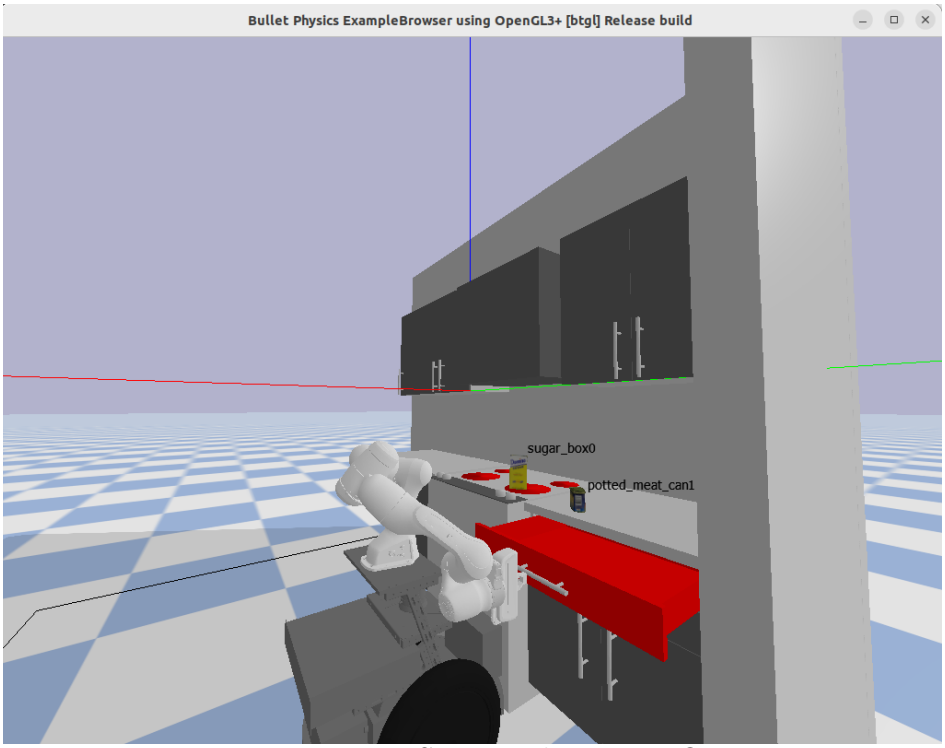

Figure 3: RRT State with Drawer Open

### 3.6 Results

A complete set of RRT-created trajectories is demonstrated in the video on the README [2]. RRT generates all trajectories except for the drawer opening sequence. The entire motion is collision-free and within joint limits, but not smooth.

# 4. Trajectory Optimization

Trajectory optimization aimed to create non-sample-based motion so that PDDL actions minimize joint angle distance, provide a smooth, collision-free trajectory, and adhere to joint angle limits.

### 4.1 Problem Definition

We follow essentially the same structure given in Example 6.7 of Russ Tedrake's Robotic Manipulation course [4]. To be explicit, we solve the following optimization problem:

$$
\begin{aligned}
\min_{\substack{\boldsymbol{\theta}(t_i)\in\mathbb{R}^7 \\ i=1,\ldots,N}} \quad & \sum_{i=2}^{N} \|\boldsymbol{\theta}(t_i) - \boldsymbol{\theta}(t_{i-1})\|^2 \\
\text{s.t.} \quad & \boldsymbol{\theta}(t_1) = \boldsymbol{\theta}_0 \\
& \boldsymbol{\theta}(t_N) = \boldsymbol{\theta}_f \\
& \boldsymbol{\theta}(t_i) \leq \boldsymbol{\theta}_{\max} \\
& \boldsymbol{\theta}(t_i) \geq \boldsymbol{\theta}_{\min}
\end{aligned} \tag{1}
$$

This problem minimizes the distance between successive joint angle commands $\boldsymbol{\theta}(t_i)$ subject to two types of constraints. The first two constraints force the initial and final joint angles to be $\boldsymbol{\theta}_0$ and $\boldsymbol{\theta}_f$, respectively. The second two constraints define the joint limits of the robot. In summary, the optimization problem minimizes the distance the robot travels in joint space from its current position to a desired final position through a predefined number of points subject to joint limits. As an important note, this formulation requires specifying the number of points the robot should travel through, essentially putting an artificial constraint on angular velocity.

## 4.2 Implementation

Our implementation relies on Drake and follows the basic mathematical program examples [1]. We define a matrix of continuous variables corresponding to the seven joints and $N$ angles on the trajectory. The start and goal angles are added as linear equality constraints, and the joint limits are added as a bounding box Constraint, detailed in the README [2].

## 4.3 Assumptions

We assume there will be no collisions throughout the trajectory, meaning no constraints were created to account for collisions (however, we still achieved a collision-free trajectory, see 4.4. Instead, collision avoidance was handled through careful planning of object placement and the drawer opening action, which adds an extra vertical motion after opening the drawer. Additionally, we assume the feasible trajectory is relatively straight; i.e., a straight-line distance can approximate the number of points needed to traverse between the start and the goal.

## 4.4 Design Decisions

We chose the simplest form of trajectory optimization because any additional complications were unnecessary to achieve a collision-free path on the specified problem. We considered adding weights to the individual joints but chose not to since joint inertia was not included in the simulation. Finally, to avoid collisions with the drawer after opening, another optimal trajectory was generated to move the gripper directly above the drawer to ensure the following action did not have any collisions 4. The only hard-coded trajectory is still the drawer opening sequence, similar to RRT.

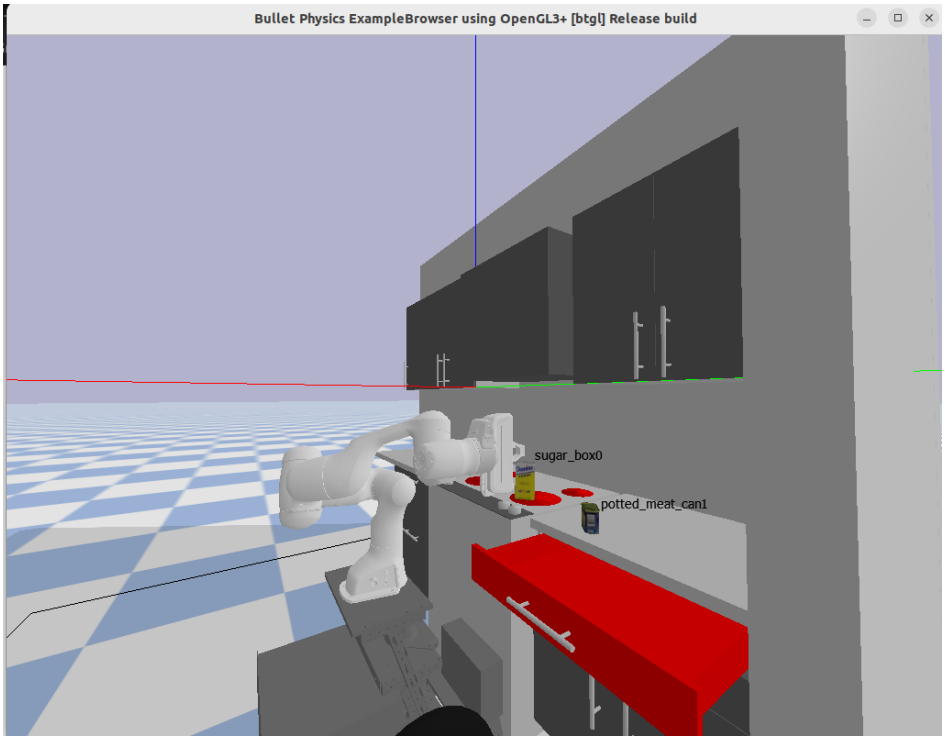

Figure 4: Trajectory Optimization State with Drawer Open

### 4.5 Solver

Our problem is convex so that we could use the basic Drake solver without modifications.

### 4.6 Challenges

We mainly encountered challenges when trying to add obstacle constraints to the optimization. Without any bounding box of the optimization given, we had to find the bounding box by trial and error with the robot. In addition, we had to use a spline instead of a specified number of points to allow for robust decision-making. With the addition of the vertical motion after opening the drawer, our formulation produced a collision-free optimization problem.

### 4.7 Results

We show smooth and collision-free trajectories between all given points for the entire PDDL sequence. See the video for empirical results [2]. The solution to the trajectory optimization problem is always the straight line path in joint space between the start and the goal.

## 5. Comparison

Please see the videos on the README [2]. RRT provided a sampling-based method and easy-to-implement algorithm to operate the robot arm in the kitchen

environment. Although producing not-optimal, non-smooth motion, RRT converged quickly, adhered to joint angle limits, and avoided collisions. On the other hand, trajectory optimization produced smooth motion and minimized angular distance while also adhering to joint angle limits. However, it did not have explicit collision avoidance constraints. This was overcome by careful arm placement after opening the drawer to ensure collision-free motion. While trajectory optimization is preferred, its lack of explicit collision avoidance constraints is a disadvantage compared to RRT in environments with many obstacles. Future work would include adding collision avoidance constraints to the optimization and developing an RRT* method to smooth the sampling-based motion.

# 6. Conclusion

We have put into practice some of the main components covered by the class: Activity Planning, Motion Planning, and Trajectory Optimization. We have successfully implemented a PDDL environment and problem that can be solved with traditional search algorithms such as Hill Climbing. Additionally, we implemented and used RRT, a sample-based algorithm that allowed us to find a set of joint angles to reach a desired objective with the robot arm. Finally, we added Trajectory Optimization to find optimal paths via a formally defined constrained optimization problem. This project meets the project's intent and satisfies all extra credit criteria.

# References


[1] pyDrake. URL https://drake.mit.edu/pydrake/.

[2] PADM Group Project README. URL https://github.com/lopenguin/principles-of-autonomy-project/tree/main.

[3] Anoop Sonar. Motion Planning. MIT Course 16.413 Principals of Autonomy and Decision Making, 2023.

[4] Russ Tedrake. Motion Planning. URL https://manipulation.csail.mit.edu/trajectories.html#section2.

[5] Brian Williams. Activity Planning as Heuristic Forward Search. MIT Course 16.413 Principals of Autonomy and Decision Making, 2023.