Motion Planning RBE 550

Project 5: Building the Two Towers

DUE: 6/11 (Init Report) , 24/11 (Interim Report), 12/12 (Final Report) at 11:59 pm.

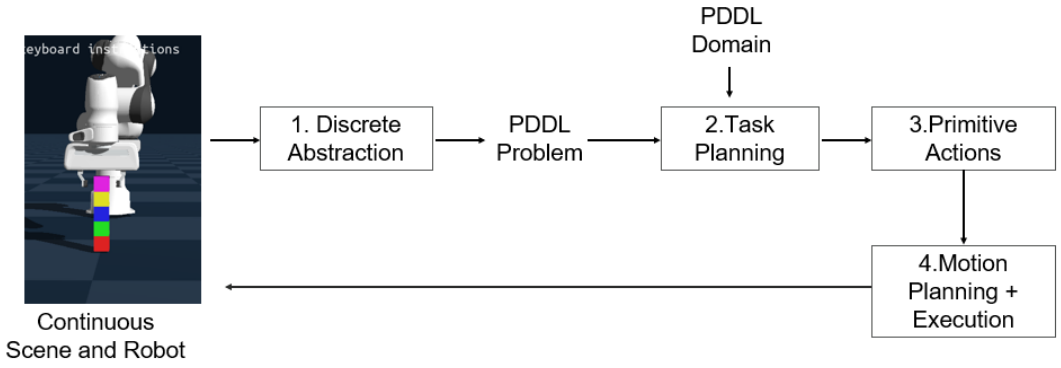


Figure 1: Overall pipeline of your project.

Modern robotic systems must reason both **symbolically** (e.g., determining what sequence of high-level actions achieves a goal) and **geometrically** (e.g., ensuring collision-free motion). In this assignment, you will build a complete **Task and Motion Planner (TAMP)** capable of manipulating blocks within the `Genesis` simulator. The overall schematic of the pipeline is shown in Figure 1. You are responsible for implementing all major subsystems described below.

In this project, you will implement a simple yet complete integration of task and motion planning. Your system should attempt to solve the motion planning problem before executing each action. The task planner should be re-invoked after every execution step to handle possible execution failures gracefully (e.g., an object slipping or a tower collapsing).

In a nutshell, your TAMP pipeline should operate as follows:

1. **Symbolic Abstraction (Lifting):** Convert the continuous state representation into a PDDL format using suitable symbolic abstraction functions that you will implement.
2. **Task Planning:** Encode the task as PDDL goals and use an existing PDDL planner to compute a high-level plan.
3. **Primitive Actions (Grounding):** Take the first high-level action from the computed task plan and formulate it as a motion planning problem—for example, by defining appropriate start states, goal configurations, and obstacles to avoid.
4. **Motion Planning and Execution:** Use `OMPL` to generate a feasible, collision-free trajectory. The provided OMPL wrapper can be used for this step. Finally, execute the resulting trajectory in `Genesis`.

Each of these subsystems is described in more detail in the following sections.

1) Symbolic Abstraction (Lifting)

Implement a symbolic abstraction pipeline that reads the state of all objects in the `Genesis` scene and determines which predicates are true or false. Given the current position of the robot q, and the positions of the blocks $b_1, b_2, \ldots, b_n$, your system should generate a set of logical predicates that describe the current scene.

For example:

`ON(A,B)`, `ONTABLE(B)`, `CLEAR(C)`, `HOLDING(A)`, `HANDEMPTY()`

The choice of predicates is up to you, but the standard `Blocksworld` PDDL domain should be sufficient to complete the first two tasks.

2) Task Planner

Next, you will integrate a **symbolic task planner**. We recommend using `pyperplan`[1], which includes a `blocksworld` example closely aligned with this assignment. Your task planner should take as input a symbolic description of the world (STRIPS-style PDDL) and output a sequence of high-level actions such as:

`PICK(A)`, `STACK(A,B)`, `PUTDOWN(A)`

3) Motion Primitives

Create motion primitives corresponding to the actions output by your task planner. Each primitive (e.g., `pick_up(x)`, `stack(x,y)`, `put_down(x)`, `unstack(x,y)`) should:

- Compute a feasible end-effector pose using inverse kinematics (IK),
- Validate the IK solution for collisions and reachability,
- Use your `OMPL` wrapper to generate a collision-free trajectory,
- Execute the trajectory in `Genesis`, controlling both arm and gripper.

You are free to design this module as you see fit—you may use direct control, motion planning, or a hybrid of both. Your implementation should be robust and capable of handling failures gracefully.

4) Motion Planning and Execution

Basic motion planning and control functionalities are already implemented for you in `planning.py` and in the built-in `Genesis` controllers. You are encouraged to explore and modify these components to better fit your pipeline design.

[1] https://github.com/aibasel/pyperplan

5) Integration and Execution Loop

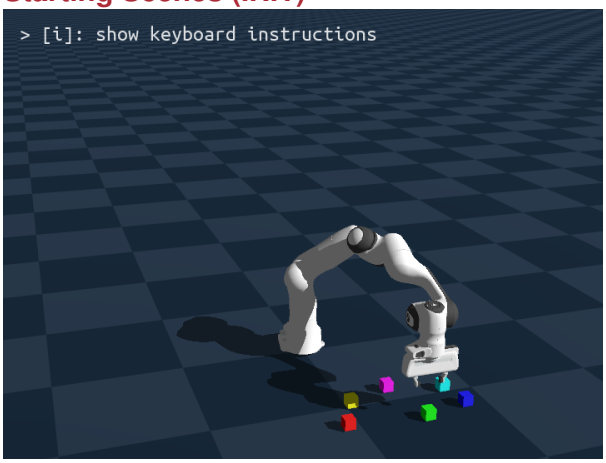
Finally, integrate all components into a single pipeline:

1. Convert the current scene into symbolic predicates.
2. Call the task planner to produce a sequence of high-level actions.
3. Execute the *first* primitive from the sequence using your motion planner.
4. Re-ground the predicates and re-plan if necessary.

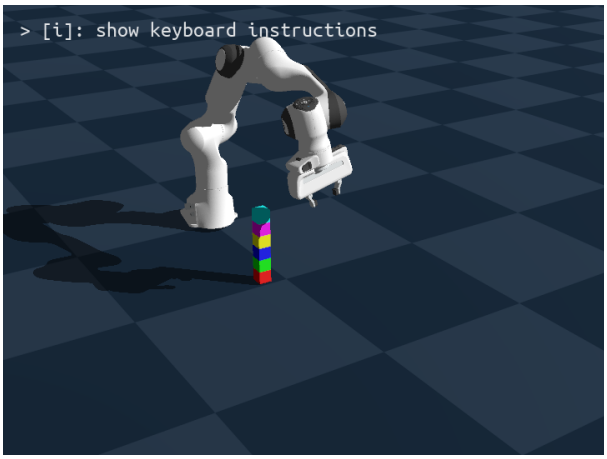
This iterative *execution–grounding–replanning* loop ensures robustness, as execution may fail (block slips from gripper) or cubes slide on top of each other

Tasks

Starting Scenes (INIT)



(a) Starting scene with all blocks arranged on the floor.



(b) Starting scene with six blocks stacked in a single tower.

Figure 2: Starting scenes used in the assignment: (a) blocks on the floor; (b) pre-built six-block tower.

You are provided with two starting scenes, each including six colored blocks. The starting scenes are initialized with small pose noise so the blocks do not start from identical positions. The first scene has all blocks on the floor, as shown in Figure 2(a). The second scene begins with a six-block tower, as shown in Figure 2(b). Do not modify these initial configurations.

The starting scenes are initialized with some noise to ensure the blocks are not starting always for from the same position. These scenes are initialized in `scenes.py` and you should not modify. You can add more init scenes for the final task but you should not change these two initial scenes.

Goal1: Build the two Towers

Given any of the two initial arrangements, your first goal is to create **two towers**, each composed of three blocks. The first tower should follow the color order (top to bottom) **RED–GREEN–BLUE**, and the second tower should follow the order **YELLOW–MAGENTA–CYAN**, as illustrated in Figure 3.

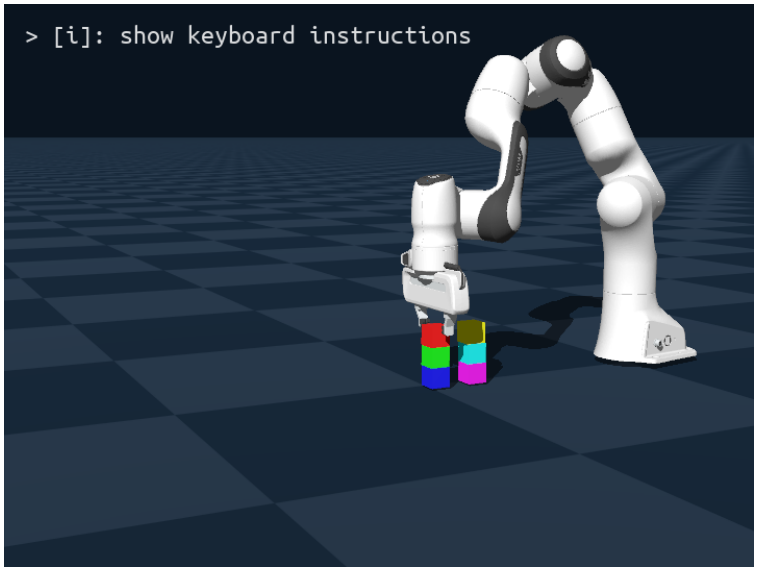


Figure 3: Target 1 configuration for the two towers: RED–GREEN–BLUE, YELLOW–MAGENTA–CYAN.

Goal2: 5-Block Tower

Starting again from both initial scenes, your second goal is to build a **5-block tower**. The order your 5-block tower follow (from top down) **MAGENTA–YELLOW–BLUE–RED –GREEN.** as shown in Figure 4.

Goal 3: Tallest Tower Challenge (Bonus)

For extra credit, attempt to build the **tallest stable tower possible**. You may introduce additional blocks into your scene if desired. For each additional block over 5 you successfully add before the structure collapses, you will receive **2 bonus points**.

You are free to choose any color order for your tower, but you must start from all blocks being on the floor, similar to the initial scene 1 Figure 2(a)

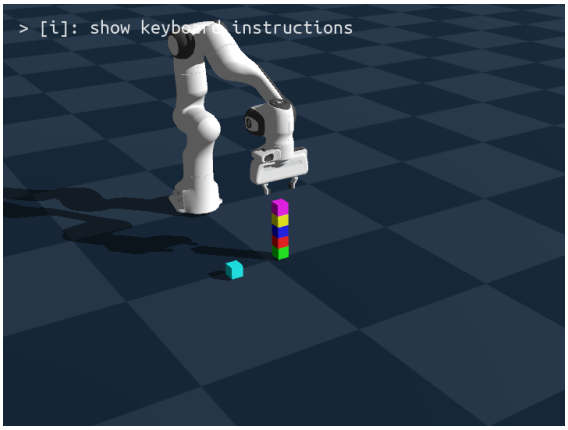


Figure 4: Example of a valid 5-block tower configuration. Taller, stable towers earn extra credit.

Goal4: Special Structure

Each team will be assigned a **special design task** to demonstrate creativity and the ability to generalize your TAMP system beyond vertical stacking. These designs will be distributed once all teams have been finalized. The will be from this manipulation competition: https://github.com/ManipulationNet/mnet_block_arrangement_example_instructions/tree/main/images

Alternatively, you may propose your *own creative design* by the project deadline. Your creative design must:

- Involve at least **nine blocks**,
- Include **non-uniform layers** (i.e., the structure should not be decomposable into repeated or homogeneous stacking tasks),
- Require different predicates from the classic blocksworld example

This part of the project is meant to showcase your team's creativity, robustness of your planning pipeline, and ability to handle complex, non-trivial assembly tasks.

Deliverables

There will be 3 Deadline to keep in mind:

1. Form group and Declare Project Topic. **Due: November 6**.
2. Interim Report. **Due: November 24**
3. Final Presentation, Final Report, and code. **Due: December12 (Late Days cannot be applied here)**

Init Report [5 points]

A singe page pdf mentioning your group members, and a screenshot of successfully running the provided genesis demo.

If you are proposing your own project you must have already talked with the advisor and have a draft approved. Your Init report should be a detailed description of your proposed project.

Interim Report[25 points]

Report: Should be compiled with the IEEE conference format and include details of your progress so far. At minimum you are expected to be able to stack 3 blocks on top of each other. It is okay if you hardcode the task plan for now if you are still working on your task planner. Your report should include:

- A brief introduction on the task and motion planning problem.
- The technical components on how you implemented each subsystem so far.
- Plan of action for rest of the components

- Biggest challenge you had to overcome so far.
- Break-down contributions of team members and total person hours spend on the project so far.

The interim report is to ensure you work on your project throughout the semester, and also an opportunity for you to receive feedback before your final submission.

Final Presentation, Report and code[70 points]

1. **Presentation:** It should be 8 min presentation +2 min Questions. It should include at minimum:
   (a) Team Members, and chosen topic.
   (b) Problem Description (In your own words, one slide).
   (c) Description on how you solved each component.
   (d) Detail description of your special design project and adopted solution
   (e) Timetable and task delegation.
   (f) Initial results you might have.

1. **Report:** Final Report including the information mentioned in the previous section
2. **Code:** A zip file containing your source code, with a README.md file explaining how to install the code, and how to run it to recreate figures and results from your report.