Python Applications for Robotics

Spring 2021

Instructor:	Zeid Kootbally	**RWA#1:**	Simulating task planning in Python
Email:	zeidk@umd.edu	**Due date:**	03/10/2021

1 Instructions

For this assignment you will need to plan a sequence of operations to be done by a robot in order to execute a manufacturing task.

- The plan is generated offline, i.e., you generate a plan first and then task the robot to follow the instructions in the plan. The assignment only focuses on plan generation and not on plan execution.
- The manufacturing task for this assignment is kitting (or kit building). Kitting is the process by which individually separate but related items are grouped, packaged, and supplied together as one unit. Frequently, these items are assembled into a kit and delivered to the assembly line for workers to integrate onto each individual final product.

We will use a dual-arm gantry robot for this assignment. Each arm consists of a suction gripper to grasp parts from the bins and release them in a kit tray (located on an automated guided vehicle (AGV)). The workcell for this assignment is represented in Figure 1.

The key features for task planning in this context are:

- Initial state: State of the world when the simulation starts.
- Goal state: State of the world that you need to reach.
- Robot actions: Actions the robot can perform to go from the initial state to the goal state.

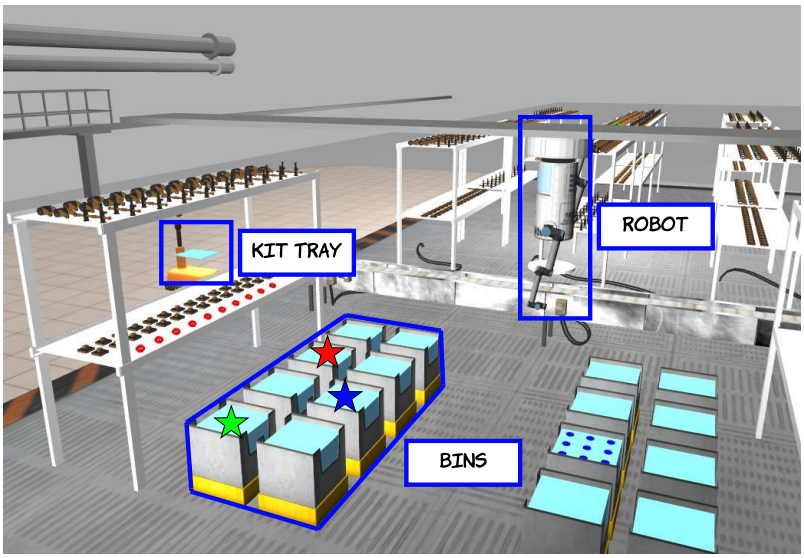

Figure 1: Snapshot of ARIAC 2020 Gazebo environment. Note: The bins depicted in this figure do not have any part but you will need to hard code how many parts each bin contains. Although not very relevant for the assignment we can expect the bin with the red star will only contain red parts, the bin with the green star will only contain green parts, and the bin with the blue star will only contain blue parts.

2 Initial State

The initial state represents the state of the world when your program starts. The initial state informs on the status of the following components.

2.1 Robot Location

The robot can be at 3 different locations in the environment:

- at_home: This is the location of the robot as shown in Figure 1.
- at_bin: The robot is located next to the bins. The robot can grasp any parts from any bins when at this location.
- at_tray: The robot is located next to the kit tray. When at this location, the robot can place parts in the kit tray.

2.2 Number of Parts in Bins

In this assignment you will use parts of the same type but of different colors (see Figure 2) and we can have a maximum of 10 parts of each color in each bin depicted in Figure 1.

Figure 2: Available part types and colors in the bins.

2.3 Gripper Status

The robot has two arms and each arm has a gripper (`left_gripper` and `right_gripper`).
The gripper has 2 states:

- `empty`: The gripper is not holding any part. In the initial state the gripper is always empty.
- `not_empty`: The gripper is holding a part.

2.4 Kit Tray Status

A kit tray is used to do kitting. The robot picks up parts from bins and places them in the kit tray.
The kit tray has 2 states:

- `complete`: The kit tray has all the parts, as specified in the goal state.
- `incomplete`: The kit tray has some parts, but not all the required parts. The kit tray is also considered incomplete when it is empty. In the initial state the kit tray is always incomplete.

3 Goal State

The goal state is the state that you need to reach by performing a series of robot actions. For this assignment, the goal is simply specified by the number of parts to place in the kit tray.

4 Create Initial and Goal States

To create the initial and the goal states you need static and dynamic information:

- Initial state:
  - Set both grippers to be `empty` (hard code this information in your program).
  - Set the robot to be at the location `at_home` (as shown in the picture).
  - Prompt the user for the number of parts in each bin.

```
How many red/green/blue parts in the bins?  9 5 2
```

    * If the user enters a number greater than 10 then you need to prompt the user to enter a new number.
  - Prompt the user for the number of parts of each color already in the kit tray.

```
How many red/green/blue parts already in the kit tray? 0 1 1
```

- Goal state:
  - Prompt the user for the number of parts of each color to place in the kit tray.

```
How many red/green/blue parts to place in the kit tray?  10 2 5
```

    * If the user enters a number of parts greater than there are actually in the bins then terminate the program with a nice message.

```
Not enough red parts for kitting: 10 needed, 9 available.
Not enough blue parts for kitting: 4 needed, 2 available
...exiting.
```

5 Robot Actions

- To go from the initial state to the goal state you will use robot actions. Each action of the robot will change the state of the environment and your program will stop when the goal state is reached.
- In other words, use robot actions to go from 0 **red** part, 1 **green** part, and 1 **blue** part in the kit tray to 10 **red** parts, 2 **green** parts, and 5 **blue** parts in the kit tray (based on the example for the goal state I provided earlier). To reach the goal state, the robot will need to pick up 10 **red** parts, 1 **green** part, and 4 **blue** parts.
- The robot can perform 4 actions: `move_to_bin`, `move_to_kit_tray`, `pick`, and `place`. Each action has a set of preconditions and a set of effects. All preconditions must be true before the action can be carried out. After executing the actions the effects are applied. Actions will be encoded as Python functions.
- When more than 1 part of the same color is needed, you need to use both arms to pick 2 parts at a time. In our example, 4 extra **blue** parts are needed in the kit tray. Your robot should be able to grab these 4 parts by doing two trips (and not four).

5.1 pick

This action allows one arm to pick up a part from a bin. Here is what the function may look like: `pick(arm: str, empty_gripper: bool, part: str)`

- Preconditions:
  - The robot is at location `at_bin`.
  - The gripper is empty.

  - There is at least one part of the specified color in the bin.
  - The kit is incomplete.
- Effects:
  - The gripper is not empty (it's holding a part).
  - Decrement the number of parts in the bin.

5.2 place

This action places a part in the kit tray. Here is what the function may look like: `place(arm: str, empty_gripper: bool, part: str)`

- Preconditions:
  - The robot is at location `at_tray`.
  - The gripper is not empty (it's holding a part).
  - The kit is incomplete.
- Effects:
  - The gripper is empty.
  - Increment the number of part of the specified color in the kit tray.

5.3 move_to_bin

This action moves the robot next to the bins. Here is what the function may look like: `move_to_bin(empty_gripper: bool)`

- Preconditions:
  - The robot is not at location `at_bin`.
  - The gripper is empty.
  - The kit is incomplete.
- Effects:
  - The robot is at location `at_bin`

5.4 move_to_tray

This action moves the robot next to the tray. Here is what the function may look like: `move_to_tray(empty_gripper: bool)`

- Preconditions:
  - The robot is not at location `at_tray`.
  - The gripper is not empty.
  - The kit is incomplete.
- Effects:
  - The robot is at location `at_tray`

6 Code Execution

- For this assignment I would highly suggest you use a dictionary and then update the dictionary from the changes that occur in the environment. For instance, set both grippers to True (for empty) when you create the dictionary and then change their values to False after picking up parts.
- When the kit is complete then output:
  - The generated plan.
  - The number of parts of each color in the kit tray.
  - The number of remaining parts in each bin.
- Below is an example of the output from the program. Please note that you do not need to have the exact format or the same sentences. However you must follow the instructions.

```
==========================================
How many red/green/blue parts in the bins?  10 10 10
How many red/green/blue parts already in the kit tray? 0 1 1
How many red/green/blue parts to place in the kit tray?  3 2 3
==========================================
Generating a plan...

====================================
---move_to_bin
---pick red part with left arm
---pick red part with right arm
====================================
---move_to_tray
---place red part with left arm
---place red part with right arm
====================================
---move_to_bin
---pick red part with left arm
====================================
---move_to_tray
---place red part with left arm
====================================
---move_to_bin
---pick green part with left arm
====================================
---move_to_tray
---place green part with left arm
====================================
---move_to_bin
---pick blue part with left arm
---pick blue part with right arm
====================================
---move_to_tray
---place blue part with left arm
---place blue part with right arm
====================================
Summary:
The kit tray has 3 red part(s) -- The bin has 7 red part(s) left
The kit tray has 2 green part(s) -- The bin has 9 green part(s) left
The kit tray has 3 blue part(s) -- The bin has 8 blue part(s) left
```

7 Python Project

For this assignment:

- You need to do it individually. You well gain lots of practice in solo mode.

- You have 2 weeks to work on it.
- Use procedural programming.
- Create a project in the format LastnameFirstname (e.g., KootballyZeid).
- In this project, create at least 1 package (**rwa1**) which consists of at least 2 modules: **taskplanning** and **taskexecution**. **taskplanning** consists of the code which does task planning (functions and data structures). **taskexecution** calls function(s) from **taskplanning** to run the program.

  It is a good idea to separate your code implementation (stored in **taskplanning**) from your code execution (stored in **taskexecution**).

  - Module **taskplanning** (`taskplanning.py`): This is where you define the initial state, the goal state, the robot actions, and the function that will run the whole project (I call it `plan()`).
  - Module **taskexecution** (`taskexecution.py`): This module imports **taskplanning** and runs your program. In the example below, the function `plan()` is defined in **taskplanning** and may call other functions for user inputs, for generating a plan, for printing out messages on the screen, etc.

    ```
    import rwa1.taskplanning

    rwa1.taskplanning.plan()
    ```
  - You have to document your code.
  - A grading rubric will be posted for this assignment.

8 Submission

Before submitting your project:

- If needed, include **instructions.txt** in your project to explain how to run your code. Otherwise we will import it in PyCharm and we will run `taskexecution.py`.
- Zip your project, it should have the following format LastnameFirstname.zip (it can be any type of compression format, not just zip).

Submit your project on ELMS.

9 Grading Rubric

Rubric	Point
Code conforms to PEP8	2
Code documented	3
Documentation provided with submission	2
Package and modules present	5
Program properly handles user inputs (see Section 4)	9
Program properly generates a plan if goal is reachable (see Section 6)	9
Total	30