Mini-Project #2: Motion Planning and Generation for a Robot Arm

Overview:

Imagine that you are helping to set up an assembly line. The section of the line you are working on uses robots to remove some packing bolts and set them aside for later use. This means that the arm must remove the bolts from the part and sort them into the appropriate bins. However, this is a reconfigurable manufacturing line - you won't always be dealing with the same part. When they change to a new part they send an excel spreadsheet that gives the coordinates where each type of bolt is located. You are responsible for creating code that translates any valid spreadsheet data into an efficient set of instructions for the robot. Efficiency is important here because the bolt removing operation is the slowest one on this line - decreases in bolt removing time will directly translate into the ability to create more finished parts each day.

We were going to simulate this task using the Mechanical Engineering Department's ST Robotics Firefly manipulator arms (see Figure 1 below), but physical testing won't be possible due to current stay-at-home restrictions. However, your mission is still to write a program that will generate a set of commands for the robot to efficiently move sets of "bolts" from their holes into storage "bins" off to the side of the part. We will define efficiency as the total distance the robot arm travels to pick and place the objects, since the time required depends on the total distance.

For this project, we will use your code to simulate a virtual robot using some basic animation code that the instructors will supply. Then, when we return to Rose next year, you can bother your instructor to run your code on the actual robot if you are interested to do so!

Figure 1. ST Robotics Firefly manipulator arm.

What Your Program Will Do:

First of all, your program will need some data. The robot, the "bolts", and the "bins" will be on a virtual table-top. **The robot always starts and ends at (0,2000)**, where the first number represents the *x*-location and the second number represents the *y*-location.

There are three types of bolts (Type 1 = red; Type 2 = green; Type 3 = blue). **There are always 3 bolts of each type on any given part**. The storage bins are arranged around the edges of the robot's work area (see Figure 2 below). The coordinates for the storage bins are given in the figure.

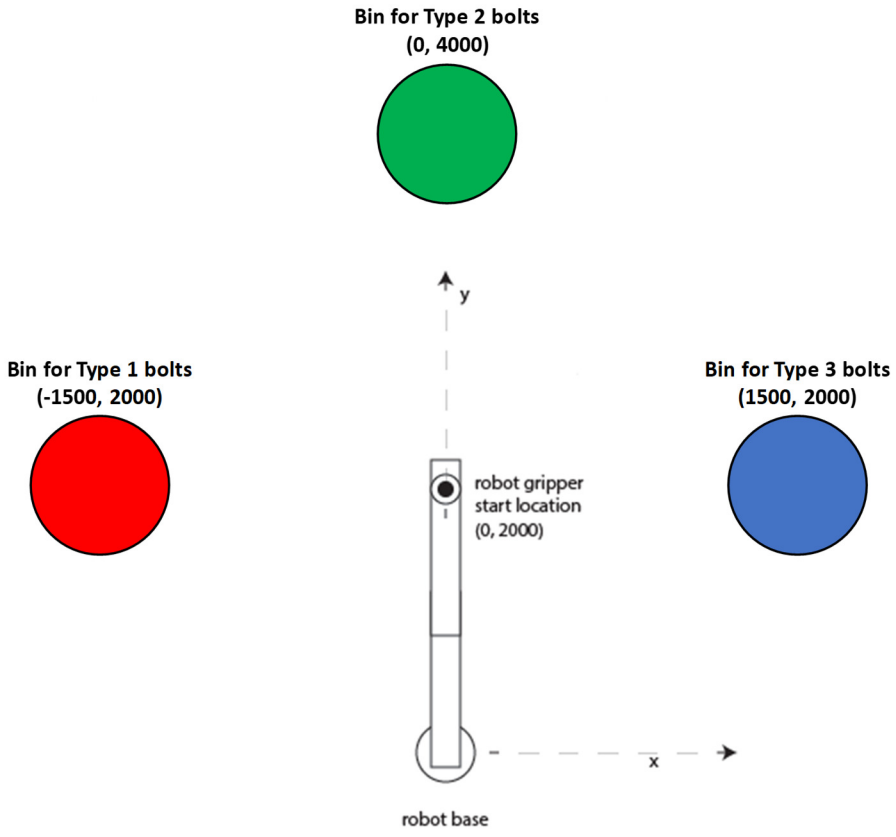

Figure 2. Layout of Robot Workspace

The *x*- and *y*-coordinates of the bolts to be removed will be given to you as an Excel spreadsheet. The table below shows the format of the spreadsheet:

Bolt 1 x	Bolt 1 y	Bolt 1 bolt type (1,2,or 3)
Bolt 2 x	Bolt 2 y	Bolt 2 bolt type (1,2,or 3)
Bolt 3 x	Bolt 3 y	Bolt 3 bolt type (1,2,or 3)
Bolt 4 x	Bolt 4 y	Bolt 4 bolt type (1,2,or 3)
…	…	…
Bolt 9 x	Bolt 9 y	Last Bolt type (1,2,or 3)

There are two sample Excel sheets posted on the course web site which you may use to test your program. You can also create your own sample data, of course.

Your program will have to plan the sequence in which the bolts are to be removed (picked up) and placed in the storage bins (put down). There are thousands of possible sequences for moving these bolts to the bins, so finding the optimal solution will be a challenge; you may just want to find a good solution instead (*i.e.* use your time wisely while working on this project).

Recall that we will use the total straight-line distance travelled between points by the robot arm as the measure of efficiency (remember: the robot will always start and finish at (0,2000), so don't forget to include those associated distances in your total).

Once your program has a set of moves planned, you will need to generate the robot language (Roboforth) code to tell the robot to make those moves. To help you out, we have written a set of Matlab functions that print out the Roboforth code to perform simple tasks. You will call those functions from your Matlab code. (The functions are on the course website and are described on the next page.)

Functions:

First, let's discuss the Matlab functions which will be provided for your use. They are basically translators, which generate a set of commands in Roboforth, the language that the robotic arm understands.

DO NOT MODIFY THESE FUNCTIONS!!! THIS COULD BREAK THE ROBOT. Just use them as we have given them to you. These functions are very simple to use.

Function	Inputs	Outputs	Purpose
`file_no = initialize('filename.txt')`	`filename` is the name of the text file for the commands.	`file_no` is the number used in subsequent commands and functions for `fprintf`.	Prepares the text file. It also moves the robot arm to the start position.
`moveto(file_no,x,y)`	`file_no` is the number of the output file obtained from the initialize function. `x,y` are the coordinates to which you want to move the arm.	None. Commands are generated and appended to the text file without sending a result back to the Matlab session.	Moves the arm from its present position to the position you have specified.
`pickup(file_no)`	`file_no` is the number of the output file obtained from the initialization function.	None. Commands are generated and appended to the text file without sending a result back to the Matlab session.	Moves the arm down, closes the grips and brings the bolt back to the plane of motion.
`putdown(file_no)`	`file_no` is the number of the output file obtained from the initialization function.	None. Commands are generated and appended to the text file without sending a result back to the Matlab session.	Moves down, "screws in the bolt" (puts it on the table), and then moves back to the plane of motion.
`close_out(file_no)`	`file_no` is the number of the output file obtained from the initialization function.	None.	Closes the output file and moves arm to the finish position.

Next, let's turn our attention to the Matlab functions you will need to write. **You'll need to create a distance function**, one which takes two points in a plane and returns the distance between them. A more advanced function, which would use the distance function, would take a sequence of points and **calculate the total travel distance associated with that sequence**.

Testing:

We have posted two sample spreadsheets on the website that you can use in developing your code. To test your output on the virtual robot (optional, but recommended), course instructors have created a function to animate a virtual robot responding to the code you provide it.

On the morning of Friday, May 8th, we will post a ***new*** spreadsheet of bolt locations; you will have until 11:59 pm on Friday, May 8th to generate your ***new*** Roboforth code and submit it to the Moode dropbox (along with your Matlab code). If your program is working correctly before Friday, this will only take a matter of minutes!

The instructors will then test all the submissions on the virtual robot and provide you with the winning time the following week.

(Reminder: Second Exam is Monday May 11th and Tuesday May 12th.)

Summary of Deliverables
(due 11:59 pm on Friday, May 8th to the Moodle dropbox):

1. Roboforth text file.
2. Well-documented Matlab code that
    a. accepts data from an Excel spreadsheet,
    b. prints a valid Roboforth program to a text file,
    c. computes the distance traveled by the robot in the text file submitted.

Grading:

The grade for the mini-project will be computed with the weights shown below:

40 pts	Robot performance
60 pts	Matlab Code (well documented and easy to follow)
100 pts	

Robot performance:

No submission	0 pts
Roboforth code submitted but does not correctly execute the task	12 pts
Roboforth code correctly executes task. Score depends on formula below.	24 -40 pts

$$\text{percent slower} = x = 100\% \times \frac{your\ distance - our\ best\ distance}{our\ best\ distance}$$

$$\text{robot performance score} = 24 + 16\, e^{-(x/10)}$$