ME 490: Midterm Project

Thomas Howell (tjh48)

Robot Actions

For the robot end effector to trace my initials ‘TH’ in space, it must execute a minimum of eight linear movements (Figure 1). Before beginning to trace my initials, however, the robot must first be moved to a starting position from which it has sufficient mobility to execute the rest of the movements. Once the robot is in the defined starting pose, it will begin to trace the letter ‘T’ by moving the end effector straight upwards, forming the base of the ‘T’. From here, the end effector will move left and then right twice as far to complete the top bar of the ‘T’. To begin tracing out the letter ‘H’, the end effector will move straight up to form the leftmost vertical line in the ‘H’. It will then move down half as far and right to trace the horizontal line. Finally, the end effector will move up and down to complete the rightmost vertical line. This movement will signify the completion of the robot task.

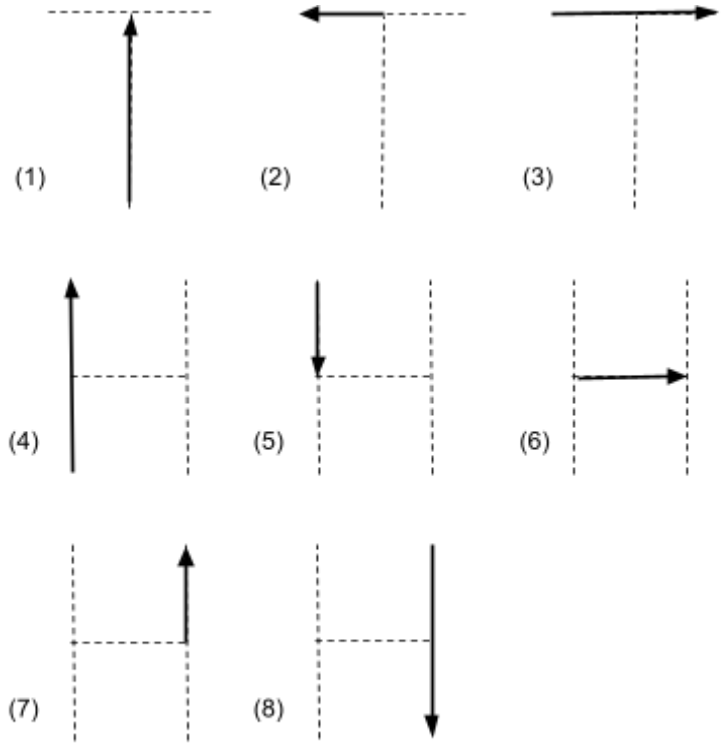

Figure 1: Robot action path for initials ‘T’ and ‘H’

Note that there is an additional movement not pictured in Figure 1 which takes place between movements 3 and 4 which is down and to the left. The purpose of this action is to reposition the end effector such that the remaining moves can be executed within the limits of the robot joints.

Code Explanation

The code to execute these movements is attached in Appendix A and can be organized into six sections. The first section (lines 1-14) is responsible for importing the required packages and initializing the objects for interfacing with MoveIt. The next section (lines 15-27) is responsible for moving the robot to its starting pose (i.e. the base of the letter ‘T’). This code defines a set of joint goals and moves the robot into the desired pose. The third section (lines 53-57) traces out the letter ‘T’ by executing movements 1-3 as described in Figure 1. The fourth section (lines 58-60) repositions the end effector to prepare the robot for tracing out the next letter. The fifth section (lines 61-67) traces out the letter ‘H’ by executing movements 4-8 as described in Figure 1. Finally, in the sixth section of code (lines 68-70), MoveIt is safely closed and the script is ended.

I define one function on lines 28-52 named **`waypoint_plan`**. The input to this function is a list of form [x, y, z] which specifies the relative change in end effector position represented in cartesian coordinates. The output of the function is a motion plan to move the end effector from its current pose to the desired position. This plan can be executed via the **`move_group.execute()`** command.

Demonstration

Below is a gif of the robotic motion. Note that the robot is already in the desired starting pose at the beginning of the gif.

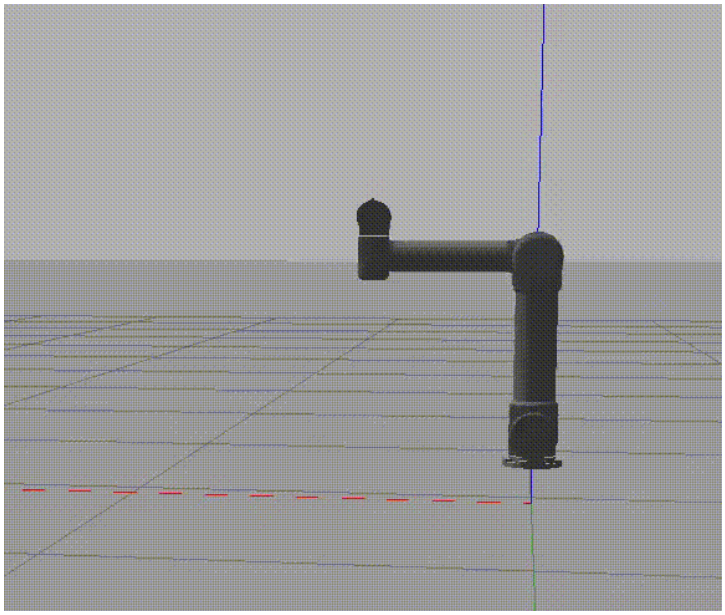

Figure 2: Robot demonstration gif

Kinematic Analysis

The DH parameters for the UR5e robot are specified in Table 1 below.

Table 1: DH Table for UR5e Robot

Link	Name	theta	a	d	alpha
1	Shoulder Pan	q_1	0.00000	0.089159	$\pi/2$
2	Shoulder Lift	q_2	-0.42500	0.00000	0
3	Elbow	q_3	-0.39225	0.00000	0
4	Wrist 1	q_4	0.00000	0.10915	$\pi/2$
5	Wrist 2	q_5	0.00000	0.09465	$-\pi/2$
6	Wrist 3	q_6	0.00000	0.0823	0

Key Poses

Several key poses are used to complete the robot task. Three of these poses are shown below where pose 1 occurs at the bottom of the ‘T’, pose 2 occurs at the left edge of the ‘T’, and pose 3 occurs at the right edge of the ‘T’.

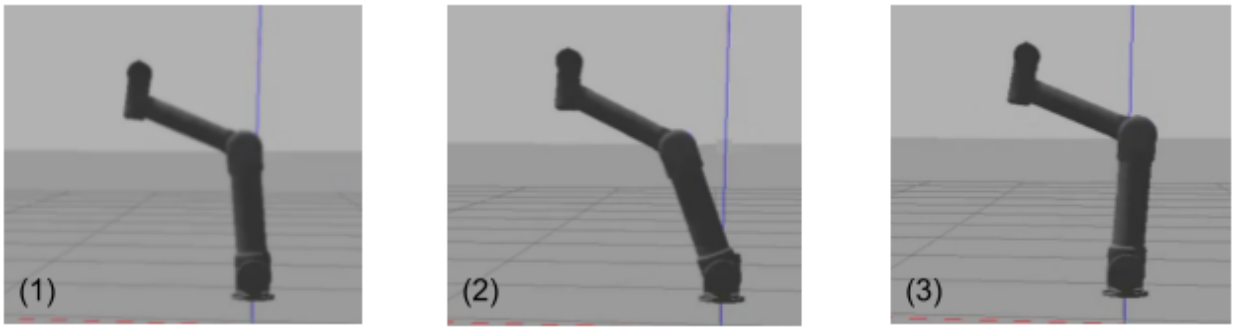

Figure 3: Three key robot poses

The joint angles in radians for poses 1-3 are listed below in Table 2. Using these joint angles, the DH parameters in Table 1, and forward kinematics, the A matrices for each pose were obtained. The Jacobian matrices were then computed from the results of forward kinematics. These calculations were performed using a Maple worksheet, which is attached at the end of this report. Since the resulting matrices are quite cumbersome to include in the body of this report, please refer to the attached Maple worksheet for the numerical results. In the worksheet, variables A1, A2, and A3 represent the A matrices for poses 1-3, respectively, and J1, J2, and J3 represent the corresponding Jacobians.

Table 2: Joint angles for poses 1-3

	q_1	q_2	q_3	q_4	q_5	q_6
Pose 1	0.0001744493 9463917564	-1.470759655 6357867	1.0706554821 144225	-2.86253537 19953655	6.0869751614 234247e-05	0.12122835812 655719
Pose 2	0.0002428004 9711578044	-1.216770073 3110234	0.7501308282 239263	-2.73156896 93354894	9.6331975886 87636e-05	0.05681715154 9830136
Pose 3	0.0005422982 942775434	-1.524489902 8690373	1.1284542295 63589,	-2.88393547 46826835	-3.1643676196 67985e-05	0.13859125794 01598

Challenges

Throughout the process of programming this robot behavior, I faced a series of challenges. One of the first issues I faced was that the robot did not always execute the movements I expected it to. I soon discovered that the root of this issue was the joint limits of the robot. Some of the movements that I prompted the robot to execute were not physically possible due to linkage interference or joint overextension. As a result, the robot appeared to be unresponsive to my commands. To resolve this problem, I identified a starting pose from which the robot had sufficient mobility to complete the task and ensured that the robot always moved to this pose before attempting to trace out my initials. Another challenge that I faced was that the robot often cut corners when provided with multiple waypoints. For the letters in my initials, it was important to achieve sharp right angles to avoid producing sloppy and unrecognizable letters. To overcome this issue, I refactored my code so that every path executed contained a single waypoint. In this way I was able to ensure that the robot hit every single waypoint while maintaining the desired angles. This method slows the robot down and requires more computation; however, it greatly improves the accuracy of the robot. For this particular task, accuracy was valued over speed.

Future Work

One aspect of robotics which I am excited to explore in the final project is computer vision. Given more time, I could expand this project in the direction of computer vision by having the robot identify and pick up a pencil in order to physically write my initials on a piece of paper. I could add an additional level of complexity by programming the robot to differentiate between various writing utensils and mediums. For example, the robot would know to only use chalk on blackboards and pencils on paper.

Takeaways

- It is very beneficial to execute robotic movements in the moveit GUI before attempting to execute them via code. The main advantage of testing out movements in the GUI is that you can verify that the motion path is contained within the task space and that it is actually achievable for the robot. If you discover that it is not, you can adjust the robot's pose and try again until you find a solution that works.

- Another takeaway I gained from this project is to make abundant use of print statements of the robot's pose, joint angles, and state. Accessing the robot's state at various breakpoints throughout your program can prove extremely useful when debugging unexpected robot behaviors. It can also provide valuable insights on the path that your robot is taking, which you can then compare against the optimal or desired path to trace back errors in your logic or programming.

Appendix A: Robot Motion Planning Code

```
1. #! /usr/bin/env python
2.
3. # Import packages
4. import sys
5. import copy
6. import rospy
7. import moveit_commander
8. from math import pi
9.
10. # Initialize node and instantiate moveit objects
11. moveit_commander.roscpp_initialize(sys.argv)
12. rospy.init_node('move_group_python_interface_tutorial',
anonymous=True)
13. move_group = moveit_commander.MoveGroupCommander("manipulator")
14.
15. # Set initial joint goals
16. joint_goal = move_group.get_current_joint_values()
17. joint_goal[0] = 0
18. joint_goal[1] = -90 * pi/180
19. joint_goal[2] = 90 * pi/180
20. joint_goal[3] = -180 * pi/180
21. joint_goal[4] = 0
22. joint_goal[5] = 0
23.
24. # Move robot to starting joint position
25. move_group.go(joint_goal, wait=True)
26. move_group.stop()
27.
28. # Define waypoint planning function
29. def waypoint_plan(wp):
30.     """
31.     Generates a cartesian path for the end effector given waypoint
parameters.
32.
33.     param wp: a list of form [x, y, z] which specifies relative change
from current pose
34.     return: a plan to move end effector to specified waypoint
```

```
    """
    waypoints = []
    wpose = move_group.get_current_pose().pose

    # Calculate waypoint from current pose and waypoint parameters
    wpose.position.x += wp[0]
    wpose.position.y += wp[1]
    wpose.position.z += wp[2]
    waypoints.append(copy.deepcopy(wpose))

    # Compute cartesian path for end effector
    (plan, fraction) = move_group.compute_cartesian_path(
        waypoints, 0.01, 0.0
    )

    # Return movement plan
    return plan

# Trace out the letter 'T' with end effector
move_group.execute(waypoint_plan([0, 0, 0.15]), wait=True) # Move in +z direction (up)
move_group.execute(waypoint_plan([0.1, 0, 0]), wait=True) # Move in +x direction (left)
move_group.execute(waypoint_plan([-0.2, 0, 0]), wait=True) # Move in -x direction (right)

# Move end effector to new starting position
move_group.execute(waypoint_plan([0.2, 0, -0.15]), wait=True) # Move in +x and -z directions (left + down)

# Trace out the letter 'H' with end effector
move_group.execute(waypoint_plan([0, 0, 0.15]), wait=True) # Move in +z direction (up)
move_group.execute(waypoint_plan([0, 0, -0.075]), wait=True) # Move in -z direction (down)
move_group.execute(waypoint_plan([-0.1, 0, 0]), wait=True) # Move in -x direction (right)
move_group.execute(waypoint_plan([0, 0, 0.075]), wait=True) # Move in +z direction (up)
```

```
66. move_group.execute(waypoint_plan([0, 0, -0.15]), wait=True) # Move in
-z direction (down)
67.
68. # Exit script
69. moveit_commander.roscpp_shutdown()
70.
```

ME 490 Midterm Project

Thomas Howell (tjh48)

Import packages

```
> with(LinearAlgebra) :
> with(ArrayTools) :
```

Forward Kinematics

```
> fk := (thet, a, d, alph) → Matrix([[cos(thet), -cos(alph)·sin(thet), sin(alph)·sin(thet), a
      ·cos(thet)],
   [sin(thet), cos(thet)·cos(alph), -sin(alph)·cos(thet), a·sin(thet)],
   [0, sin(alph), cos(alph), d],
   [0, 0, 0, 1]]) :
```

Pose 1 - Joint Angles

```
> JA1 := [0.00017444939463917564, -1.4707596556357867, 1.0706554821144225,
      -2.8625353719953655, 6.0869751614234247e-05, 0.12122835812655719] :
```

Calculate individual A matrices

```
> A11 := fk(JA1[1], 0, 0.089159, Pi/2) :
> A12 := fk(JA1[2], -0.425, 0, 0) :
> A13 := fk(JA1[3], -0.39225, 0, 0) :
> A14 := fk(JA1[4], 0, 0.10915, Pi/2) :
> A15 := fk(JA1[5], 0, 0.09465, -Pi/2) :
> A16 := fk(JA1[6], 0, 0.0823, 0) :
```

Calculate final A matrix

```
> A1 := evalf(A11 · A12 · A13 · A14 · A15 · A16, 4)
```

$$A1 := \begin{bmatrix} -1. & 0.00018150000000000000 & 0.00023490000000000000 & -0.392200000000000 \\ -0.00023490000000000000 & 7.39300000000000 \times 10^{-6} & -1. & -0.191500000000000 \\ -0.00018150000000000000 & -1. & -7.35000000000000 \times 10^{-6} & 0.758800000000000 \\ 0. & 0. & 0. & 1. \end{bmatrix} \quad (1)$$

Pose 2 - Joint Angles

```
> JA2 := [0.00024280049711578044, -1.2167700733110234, 0.7501308282239263,
      -2.7315689693354894, 9.633197588687636e-05, 0.056817151549830136] :
```

Calculate individual A matrices

```
> A21 := fk(JA2[1], 0, 0.089159, Pi/2) :
> A22 := fk(JA2[2], -0.425, 0, 0) :
> A23 := fk(JA2[3], -0.39225, 0, 0) :
```

```
> A24 := fk(JA2[4], 0, 0.10915, Pi/2):
> A25 := fk(JA2[5], 0, 0.09465, -Pi/2):
> A26 := fk(JA2[6], 0, 0.0823, 0):
```

Calculate final A matrix

```
> A2 := evalf(A21 • A22 • A23 • A24 • A25 • A26, 4)
```

$$A2 := \begin{bmatrix} -1. & 0.000201600000000000 & 0.000339000000000000 & -0.492200000000000 \\ -0.000339000000000000 & 5.51900000000000 \times 10^{-6} & -1. & -0.191600000000000 \\ -0.000201600000000000 & -1. & -5.45100000000000 \times 10^{-6} & 0.758800000000000 \\ 0. & 0. & 0. & 1. \end{bmatrix} \quad (2)$$

Pose 3 - Joint Angles

```
> JA3 := [0.0005422982942775434, -1.5244899028690373, 1.128454229563589,
          -2.8839354746826835, -3.164367619667985e-05, 0.1385912579401598]:
```

Calculate individual A matrices

```
> A31 := fk(JA3[1], 0, 0.089159, Pi/2):
> A32 := fk(JA3[2], -0.425, 0, 0):
> A33 := fk(JA3[3], -0.39225, 0, 0):
> A34 := fk(JA3[4], 0, 0.10915, Pi/2):
> A35 := fk(JA3[5], 0, 0.09465, -Pi/2):
> A36 := fk(JA3[6], 0, 0.0823, 0):
```

Calculate final A matrix

```
> A3 := evalf(A31 • A32 • A33 • A34 • A35 • A36, 4)
```

$$A3 := \begin{bmatrix} -1. & 0.000212800000000000 & 0.000511000000000000 & -0.368400000000000 \\ -0.000511000000000000 & -4.25600000000000 \times 10^{-6} & -1. & -0.191600000000000 \\ -0.000212800000000000 & -1. & 4.36500000000000 \times 10^{-6} & 0.758800000000000 \\ 0. & 0. & 0. & 1. \end{bmatrix} \quad (3)$$

Jacobian Matrix

```
> jm := (Z, On, Oi)→Concatenate(1, Z &x (On − Oi), Z):
```

Pose 1

Calculate individual Jacobian components

```
> J11 := jm(Vector(3, shape=unit[3]), A1[1..3, 4], Vector(3)):
> J12 := jm(A11[1..3, 3], A1[1..3, 4], A11[1..3, 4]):
> J13 := jm(Matrix(A11 • A12)[1..3, 3], A1[1..3, 4], Matrix(A11 • A12)[1..3, 4]):
> J14 := jm(Matrix(A11 • A12 • A13)[1..3, 3], A1[1..3, 4], Matrix(A11 • A12 • A13)[1..3, 4]):
> J15 := jm(Matrix(A11 • A12 • A13 • A14)[1..3, 3], A1[1..3, 4], Matrix(A11 • A12 • A13
          • A14)[1..3, 4]):
> J16 := jm(Matrix(A11 • A12 • A13 • A14 • A15)[1..3, 3], A1[1..3, 4], Matrix(A11 • A12 • A13
```

$\bullet A14 \bullet A15)[1..3,4])$:

Calculate Jacobian matrix

> $J1 := Concatenate(2, J11, J12, J13, J14, J15, J16)$

$$J1 := [[0.191500000000000, -0.669640989821457, -0.246765782249160, -0.0939788040169854, 0.0816794945985574, -0.000219864216295595], [-0.392200000000000, -0.000116818466446662, -0.000430481417536040, -0.000163945456306113, 0.000637220155186354, -5.51292002297268 \times 10^{-9}], [0, -0.392233401097454, -0.349788691826070, 0.0114815686321962, -0.00993562441248723, 0.000474668364670646], [0, 0.0001744493937, 0.000174449393700000, 0.000174449393700000, 0.120751503543461, 0.000234873745804501], [0, -0.9999999848, -0.999999984800000, -0.999999984800000, 0.000210650269017086, -0.999999972359008], [1, 0, 0., 0., 0.992682766002703, -7.35011413411621 \times 10^{-6}]] \quad (4)$$

Pose 2

Calculate individual Jacobian components

> $J21 := jm(Vector(3, shape = unit[3]), A2[1..3,4], Vector(3))$:

> $J22 := jm(A21[1..3,3], A2[1..3,4], A21[1..3,4])$:

> $J23 := jm(Matrix(A21 \bullet A22)[1..3,3], A2[1..3,4], Matrix(A21 \bullet A22)[1..3,4])$:

> $J24 := jm(Matrix(A21 \bullet A22 \bullet A23)[1..3,3], A2[1..3,4], Matrix(A21 \bullet A22 \bullet A23)[1..3,4])$:

> $J25 := jm(Matrix(A21 \bullet A22 \bullet A23 \bullet A24)[1..3,3], A2[1..3,4], Matrix(A21 \bullet A22 \bullet A23 \bullet A24)[1..3,4])$:

> $J26 := jm(Matrix(A21 \bullet A22 \bullet A23 \bullet A24 \bullet A25)[1..3,3], A2[1..3,4], Matrix(A21 \bullet A22 \bullet A23 \bullet A24 \bullet A25)[1..3,4])$:

Calculate Jacobian matrix

> $J2 := Concatenate(2, J21, J22, J23, J24, J25, J26)$

$$J2 := [[0.191600000000000, -0.669640980245590, -0.270997575705571, -0.0945292520034671, 0.0821985616595387, -0.000313549087339527], [-0.492200000000000, -0.000162589166071403, -0.000657983473848647, -0.000229517498271394, 0.000663279125096968, -1.08481817480002 \times 10^{-8}], [0, -0.492246506054885, -0.344908708461945, 0.00540399663773314, -0.00465869699444248, 0.000402777623273586], [0, 0.0002428004947, 0.000242800494700000, 0.000242800494700000, 0.0565853181857282, 0.000338978120232113], [0, -0.9999999705, -0.999999970500000, -0.999999970500000, 0.000137389436535505, -0.999999942548024], [1, 0, 0., 0., 0.998397767173580, -5.45097565951153 \times 10^{-6}]] \quad (5)$$

Pose 3

Calculate individual Jacobian components

```
> J31 := jm(Vector(3, shape = unit[3]), A3[1..3, 4], Vector(3)) :
> J32 := jm(A31[1..3, 3], A3[1..3, 4], A31[1..3, 4]) :
> J33 := jm(Matrix(A31 • A32)[1..3, 3], A3[1..3, 4], Matrix(A31 • A32)[1..3, 4]) :
> J34 := jm(Matrix(A31 • A32 • A33)[1..3, 3], A3[1..3, 4], Matrix(A31 • A32 • A33)[1..3, 4]) :
> J35 := jm(Matrix(A31 • A32 • A33 • A34)[1..3, 3], A3[1..3, 4], Matrix(A31 • A32 • A33
      • A34)[1..3, 4]) :
> J36 := jm(Matrix(A31 • A32 • A33 • A34 • A35)[1..3, 3], A3[1..3, 4], Matrix(A31 • A32 • A33
      • A34 • A35)[1..3, 4]) :
```

Calculate Jacobian matrix

```
> J3 := Concatenate(2, J31, J32, J33, J34, J35, J36)
```

$$J3 := \begin{bmatrix} [0.191600000000000, -0.669640901562773, -0.245096543070803, \\ -0.0937806736151728, 0.0814639459875060, -0.000350911240707938], \\ [-0.368400000000000, -0.000363145154280896, -0.000132915450265126, \\ -0.000050857104321241 6, 0.000041729703614381 1, -1.79080477120594 \times 10^{-8}], \\ [0, -0.368503850193291, -0.348830652661571, 0.0130582300106631, \\ -0.0113453682858245, 5.04322261450469 \times 10^{-6}], \\ [0, 0.0005422982677, 0.000542298267700000, 0.000542298267700000, \\ 0.137937270858674, 0.000510957078879013], \\ [0, -0.9999998530, -0.999999853000000, -0.999999853000000, \\ 0.000074803154033988 4, -0.999999869496275], \\ [1, 0, 0., 0., 0.990440964216957, 4.36484297521613 \times 10^{-6}]] \end{bmatrix} \quad (6)$$

```
>
```