

ENMT482

Robotics

Assignment 2

UR5 Manipulator Assignment

Students

Jaime Sequeira	42408823	jfs73@uclive.ac.nz
Zoren Dela Cruz	986724643	zjd15@uclive.ac.nz

2020

University of Canterbury

Table of Contents

1	Introduction.....	1
2	Frame Assignment and Rotation Matrices.....	1
2.1	Items on Table.....	1
2.1.1	Coffee Machine.....	2
2.1.2	Grinder Machine	2
2.1.3	Scraper and Tamper Stand	2
2.1.4	Cup Stack.....	3
3	Homogenous Transforms.....	3
3.1	Tool Transforms	3
3.2	Portafilter Tool	4
3.3	Movement Transforms	4
4	Path Planning and Obstacle Avoidance	5
5	Source Code and Programming	5
6	Time and Motion Results.....	6
7	Takeaways from Assignment.....	8
8	Recommendations.....	9
9.	Appendix.....	A
A.	Frame Assignments	A
B.	Tool Frame Assignments	B
C.	Homogenous Transforms Calculations	C
C.1	Grinder Tool.....	C
C.2	Cup Tool.....	C
D.	Source Code	D

1 Introduction

Industrial robots are programmable, automatically controlled, multipurpose manipulators in three or more axes. These robots are often used to aid in production tasks such as assembly, product inspection and testing. In this assignment, a UR5 robot will brew a coffee by operating multiple pieces of equipment as shown in Figure 1.

This report will describe the methodology used to create a program to brew a cup of coffee with the UR5 robot through frame assignments and transformation matrices. It will then discuss the path planning and obstacle avoidance the group implemented. The report will conclude with a time and motion analysis, recommendations for future students and possible improvements for this assignment.

2 Frame Assignment and Rotation Matrices

Reference frames and target points on parts of the machines on the table are shown in Figure 1. This section will discuss how and why arbitrary frames are set and configured. It will also demonstrate how the rotation matrices are derived.

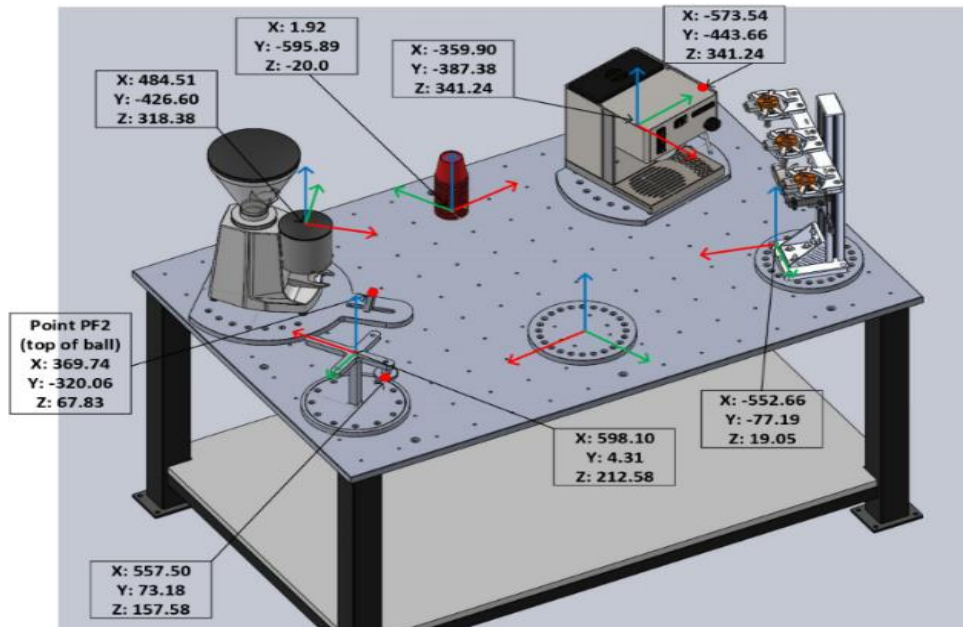


Figure 1: Layout of items on table showing reference frames

2.1 Items on Table

The first step taken in this project was to calculate the rotation matrices for each equipment's reference frame shown in Figure 1. The rotation matrices were calculated using the function *convertToRM* as seen in Appendix D. The inputs to this function were the two coordinate points on each item on the table shown in Figure 1, angle-offset and operator. The angle-offset and operator were calculated for each equipment reference frame as the *arctan* function was not used to estimate the angle. The rotation matrix equation used for the function *convertToRM* is in Equation 1.

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

After calculating the rotation matrices for each item on the table, the local coordinates provided for positions on the equipment could now be mapped to the global coordinates.

2.1.1 Coffee Machine

In this project, the robot had to push one of the buttons on the coffee machine with the grinder tool to run the machine. A cup was also placed on the coffee machine's drip tray using the cup tool. Therefore, the coffee machine had two tool reference frames associated with it. The frames assigned in the coffee machine for the tools can be seen in Appendix A.

The z-axis of the grinder tool frame was used to locate the switch on the coffee machine and was oriented perpendicular to the switch. This would allow for linear movement of the robot to turn the switch on and off. The orientation of the x and y-axis did not matter, as the part of the end-effector (grinder tool) that contacted the switch was circular and flat.

The orientation of the frame used to locate the centre of the drip tray was dictated by the position of the portafilter tool when it was locked into the coffee machine. As shown in Appendix A, aligning the z-axis of the cup tool with the y-axis of the coffee machine avoided collisions between the robot and the portafilter tool as the cup tool approached the drip tray from the side of the coffee machine. The rotation matrices of these two frames are defined in Equation 2 and 3.

$$\begin{matrix} \text{Coffee Machine} \\ \text{On Button} \end{matrix} R_{Grinder Tool} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (2)$$

$$\begin{matrix} \text{Coffee Machine} \\ \text{Drip Tray Center} \end{matrix} R_{Cup Tool} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad (3)$$

2.1.2 Grinder Machine

The grinder machine was used for grinding the coffee beans. Using the lever attached to the bottom of the grinder, the coffee beans were then dispensed into the portafilter. The reference frames assigned in the grinder machine for the tools can be seen in Appendix A. The reference frame orientations for the portafilter stand and the lever were aligned with the grinder's reference frame. The z-axis of the portafilter tool was aligned with the negative x-axis of the grinder's reference frame. These frame orientations allowed linear motions of the robot to place the portafilter tool into the stand. To achieve the same linear motion to turn the grinder on and off, the grinder tool's z-axis was aligned with the negative y-axis of the grinder's reference frame. The rotation matrices for these frames are defined in Equation 4, 5 and 6.

$$\begin{matrix} \text{Grinder} \\ \text{Portafilter Stand} \end{matrix} R_{Portafilter Tool} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (4)$$

$$\begin{matrix} \text{Grinder} \\ \text{Depositing Lever} \end{matrix} R_{Grinder Tool} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (5)$$

$$\begin{matrix} \text{Grinder} \\ \text{Grinder Power Button} \end{matrix} R_{Grinder Tool} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \quad (6)$$

2.1.3 Scraper and Tamper Stand

The scraper and tamper stand is used to remove the excess coffee and to compress the coffee grinds into the portafilter respectively. The orientations of the frames were dictated by the portafilter tool. The portafilter tool had to stay parallel to the table throughout the scraping and tamping process to avoid dropping coffee grinds.

The frames for the scraper and tamper were oriented to allow linear movement in the z-axis of the portafilter tool as shown in Appendix A. This allowed for easy control of the robot in software. The rotation matrices for the portafilter tool with respect to the scraper and tamper are defined in Equation 7 and 8 respectively.

$$\begin{matrix} \text{Tamper and Scraper Stand} \\ \text{Scraper} \end{matrix} R_{\text{Portafilter Tool}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad (7)$$

$$\begin{matrix} \text{Tamper and Scraper Stand} \\ \text{Tamper} \end{matrix} R_{\text{Portafilter Tool}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \quad (8)$$

2.1.4 Cup Stack

In order to brew the coffee, a disposable cup needed to be collected from the cup stack using the cup tool. When collecting the cup, the cup tool orientation needed to be reversed as shown in Appendix A. In Appendix A, it shows the z-axis of the cup tool was aligned with the positive x-axis of the cup stack's reference frame. The x-axis of the cup tool was aligned with the negative z-axis of the cup stack. These frames allowed the cup tool to move linearly and collect the cup. The rotation matrix of the cup tool with respect to the cup stack is in Equation 9.

$$\begin{matrix} \text{Cup Stack} \\ \text{Cup Rim} \end{matrix} R_{\text{Cup Tool}} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix} \quad (9)$$

3 Homogenous Transforms

Homogenous transforms can be used to describe a reference frame's position and orientation. In this project, homogenous transform multiplications were utilised to calculate the desired transforms. Given that homogenous transform calculations were repeated throughout the program, the group decided to create a python module to handle homogenous transform conversions to add flexibility and reduce potential calculation errors.

The function for the homogenous transform conversion was called *convertToHT* where its inputs were a rotation matrix and a position vector. This function appends the position column vector to the 3x3 rotation matrix. The function then appends the [0, 0, 0, 1] row vector to the bottom row to create the 4x4 homogenous transform.

3.1 Tool Transforms

The tools used in the coffee-making process also required assigning frames and rotation matrices to create homogeneous transforms. As seen in Appendix B, the master tool's reference frame was rotated around the Z-axis of the UR5 base frame by 50 degrees. The rotation matrix for the master tool is defined in Equation 10.

$$R_{\text{Master Tool}} = \begin{bmatrix} \cos(50) & \sin(50) & 0 \\ -\sin(50) & \cos(50) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (10)$$

Appendix B shows the rotation of the master tool around the Z-axis. The rotation of the master tool was treated as a roll change for future rotation matrices as opposed to pitch and yaw.

3.2 Portafilter Tool

The portafilter tool required two reference frames to be assigned. The first reference frame was located on the bottom edge of the portafilter tool which sits in the stand of the grinder machine. The second reference frame was placed in the centre of portafilter's cup. This reference frame was used when moving the portafilter tool with coffee grinds inside the portafilter cup. The z-axis of the portafilter tool was aligned with the x-axis of the UR5 base. This kept the portafilter tool parallel to the table and avoided spilling coffee grinds. These rotation matrices are defined in Equation 11 and 12. In Equation 12, θ represents the rotation around the portafilter tool's y-axis due to the angle of the portafilter tool's cup.

$${}^{Master\ Tool}R_{Portafilter\ Tool\ (Edge)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (11)$$

$${}^{Portafilter\ Tool}R_{Portafilter\ Tool\ (Centre)} = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad (12)$$

The reference frame at the bottom edge of the portafilter tool required compensating for the master tool's roll. In Appendix B, it shows the reference frame in the centre of the cup of the portafilter tool required a rotation matrix that incorporated both its pitch and the master tools roll. This was achieved by multiplying the individual rotation matrices together as shown in Equation 13.

$${}^{Master\ Tool}R_{Portafilter\ Tool\ (Centre)} = R_{Portafilter\ Tool\ (Centre)} \times R_{Master\ Tool} \quad (13)$$

Equations 14 and 15 define the homogeneous transforms for the respective tool ends.

$${}^{Master\ Tool}_{Portafilter\ Tool\ (Edge)}T = \begin{bmatrix} {}^{Master\ Tool}R_{Portafilter\ Tool\ (Edge)}(3 \times 3) & \begin{bmatrix} -32.0 \\ 0 \\ 27.56 \end{bmatrix} \\ 0 & 1 \end{bmatrix} \quad (14)$$

$${}^{Master\ Tool}_{Portafilter\ Tool\ (Centre)}T = \begin{bmatrix} {}^{Portafilter\ Tool}R_{Portafilter\ Tool\ (Centre)}(3 \times 3) & \begin{bmatrix} 4.71 \\ 0 \\ 144.76 \end{bmatrix} \\ 0 & 1 \end{bmatrix} \quad (15)$$

The process shown above is repeated to find the homogeneous transforms for the grinder tool, and the cup tool as shown in Appendix C.

3.3 Movement Transforms

The homogeneous transforms were used to calculate the transform relating the TCP of the selected tool to the UR5 base for different end effector configurations. The general formula to calculate this transform is shown in Equation 16.

$${}^{UR5}_{TCP}T = {}^{UR5}_{Machine}T \cdot {}^{Point\ on\ Machine}_{Machine}T \cdot {}^{Tool}_{TCP}T \quad (16)$$

However, the ${}^{Tool}_{TCP}T$ transform was unknown. The inverse of this transform was known; thus, it could be substituted into Equation 16. The equation of transforms is now fully defined in Equation 17.

$${}^{UR5}_{TCP}T = {}^{UR5}_{Machine}T \cdot {}^{Point\ on\ Machine}_{Machine}T \cdot ({}^{TCP}_{Tool}T)^{-1} \quad (17)$$

The homogenous transforms were required for all the items on the table. In the Python code, a class was used to handle all the items on the table and assign them their relevant homogeneous transforms. For each item, an instance of the Machine class was created. Each object was instantiated with a name and the homogeneous transform of the object relative to the UR5 Base frame.

4 Path Planning and Obstacle Avoidance

The goal of path planning is to generate reference inputs to the motion control system. With path planning, the best route tends to be subjective where you need to choose between the shortest or the quickest route. Before path planning, the target points such as buttons and levers were calculated. The group then moved the UR5 robot by working in the Cartesian and joint space.

After calculating the target points in the global coordinate frame, the robot was simulated in RoboDK to check if potential collisions would occur. If a collision was about to occur, the group manually moved the robot into a free space to avoid the obstacle and configured the free space as a waypoint to reach the target point. For the majority of the UR5 robot's movement, the group worked in joint space. Working in joint space avoided potential singularities. The joint angle waypoints were acquired by manually manipulating the robot and copying the joint angles into the program as a waypoint.

The Cartesian space was used for linear movements for the end-effector. The Cartesian space was used in tasks such as pressing buttons and scraping the portafilter tool as the movement for the end effector was linear which had a small chance of hitting singularities.

In this project, the group also used the Inverse Kinematics built into RoboDK to optimise the path of the robot. The Inverse Kinematics output multiple joint configurations for a certain end-effector position. The group selected the joint configuration of the UR5 by choosing configurations that would reduce the chance of collisions and minimised the joint movements to reduce the robot's run time. The group tried their best to optimise the robot's path by selecting the most optimal waypoints and joint configurations to minimise robot's run time.

One of the unique movements performed in the project was pulling the lever in the grinding machine to deposit the ground coffee into the portafilter. This movement required a circular rotation using the RoboDK API function *robot.MoveC*; however, this movement failed to compile outputting a 'Target Reach Error' code. The inverse kinematics were un-solvable with the chosen targets. An alternative method the group used was to use two linear motions to achieve a similar trajectory.

5 Source Code and Programming

In this assignment, modularity was prioritised for ease of the development process. The first module created was a script to call Rodney's function along with the required pause function shown in Appendix D. By creating this module, the program was less prone to errors when calling Rodney's functions as the required blocking statements and pause functions were called automatically.

Addition to that, each task was assigned its own script file. This made debugging the tasks easy. If the task was not behaving as expected or needed calibration, the source file for the task was debugged. Using this method also simplified the main function of the coffee-making

program. The tasks were imported into the main script and appended into a list. The main function iterated through the list calling each task's *runTask()* function. Figure 2 shows how this was implemented in the main python file run by RoboDK.

```
# Create an array of all the tasks
taskArray = [TaskA, TaskB, TaskC, TaskD, TaskE, TaskF, TaskG, TaskH, TaskI, TaskJ]

"""
-----
    And that is all there is to it. Call the main function to start making a fresh
    brew. xoxo
-----
"""

def main():

    # Send the robot to the starting position
    goHome()

    # Iterate through the task array and call the runTask function
    for task in taskArray:
        task.runTask()

# Run the python script
main()
```

Figure 2: The main python script run by RoboDK

Lastly, a class was created to parameterise the machines and items on the table. The class consisted of the machine name and the machine homogeneous transform relative to the UR5 base. Creating this class allowed the machine homogenous transform to be called in any module by importing the instantiated machine or item class.

6 Time and Motion Results

Using an industrial robot, simple tasks such as coffee brewing could be automated. Restaurants across the world are exploring options of replacing workers with robots to define themselves over their competition while achieving the same functionality. In this section, it will evaluate the advantages and limitations of using a UR5 robot to brew coffee in cafes.

In this project, the simulated robot required 9 minutes to brew one coffee. If the robot were to operate continuously in 8-hour days, it would be able to make 53 coffees. Applying a costing analysis, the total cost to use a UR5 robot and its total revenue in one year to evaluate the breakeven point is shown in Figure 3. Table 1 summarises the fixed and variable cost of the UR5 and profit for every coffee sold.

Table 1: Costing summary

Robot Cost	Tool Cost	Robot Maintenance	Cost of Coffee
\$40, 000	\$5, 000	\$400 per annum	\$4.50 per Coffee

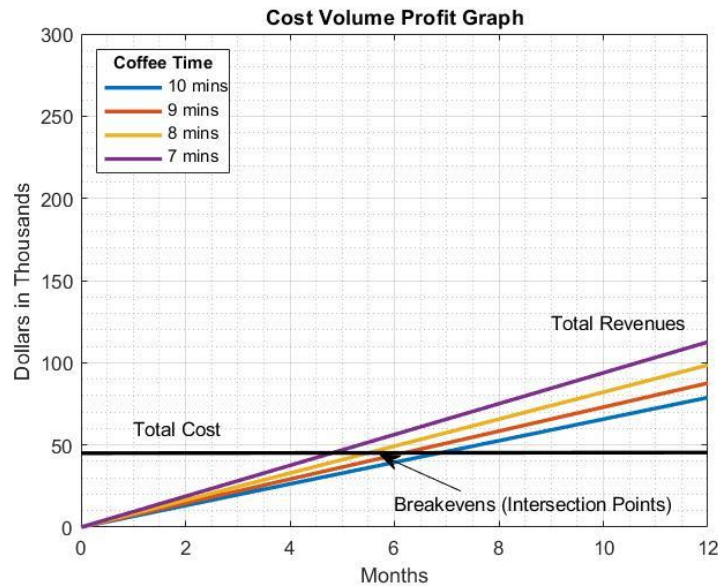


Figure 3: Breakeven/Costing analysis

As shown in Figure 3, the breakeven point if the robot makes a coffee every 9 minutes is approximately 6 months. Overall, in a single year using a UR5 robot, the annual profit is approximately \$42,000. The breakeven time can be shortened if the robot's performance was optimised. Shown in Figure 3, if the robot can make a cup of coffee every 7 minutes, the breakeven point is approximately 5 months. With that, the UR5 robot would be able to make maximum of 68 coffees a day which would lead to a potential annual profit of \$66,690.

From Rodney's fantastic human making coffee video, it takes approximately 52 seconds to brew a single coffee. Comparing the UR5 to an employee, an employee would be able to make 553 coffees in an 8-hour working time (disregarding breaks). The annual profit is \$819,607 per employee. This trumps the UR5's profitability; however, this is assuming that the employee makes the coffees non-stop and every coffee is sold. In reality, this would not be the case as from typical cafes sell 230 coffee cups per day.

Directly comparing the annual profit from a UR5 to an employee, the employee brings in more profit with their ability to make coffee in a shorter time. However, not every coffee made in an 8-hour span would be sold. But, if a coffee shop sells a maximum of 230 coffee cups per day, an employee is still more profitable as its total revenue in a year is much higher. With that, there are several limitations with using a UR5 robot compared to an employee.

The limitations of using a UR5 robot is that it cannot handle customer demands. With the UR5 being programmed to perform a specific task at a high level of accuracy, special requests from customers may not be serviced. If a customer wants their coffee brewed in a different way, the UR5 robot will need to be reprogrammed to achieve the customer's desire. This will require re-calibration of the robot and will be a setback during production. It will be more beneficial for an employee to serve special requests from customers for orders that the UR5 cannot perform.

Another limitation of using the UR5 robot is that it is unable to multi-task in comparison to an employee. At peak hours, multiple coffees will be requested in which the UR5 robot would have difficulty to adjust. With the UR5 robot, it can only make a single coffee at once. With an

employee, they would be able to multi-task to prepare multiple coffees at once rather than making them one at a time. However, a potential solution to this peak hour problem is to have additional UR5 robots in the shop. Considering the area each of these UR5 robots requires to operate, it would not be feasible.

A disadvantage of using a UR5 robot in a coffee shop is that system is not ideal for dynamic environments. With potentially changing environments, if the UR5 robot is not encaged, the robot would fail to perform at its best. If it collides with a dynamically changing obstacle such as an employee's arm, the robot would enter an emergency stop and would need to be restarted again. However, if the robot was encaged, the additional cost of the cage needs to be added to the total cost, which previously was \$45,000. The last disadvantage the UR5 offers is the cost of additional overhead cost due to its power consumption.

The advantage of using a UR5 robot is that it is programmed to complete the task with a high level of accuracy. This reduces the potential of failed brewed coffee as no human errors could possibly occur. Another advantage is the prestigious effect of having a robot brewing coffee in a coffee shop. This robot could be an attraction to potential customers that would drive the sales for its unique method of making coffee.

Overall, the UR5 robot has its positives and negatives. Using a UR5 robot is prestigious and can be used as a marketing tool to boost the brand image for your coffee shop. However, there are limitations when it comes to being able to meet the demands and additional overhead cost when operating this robot. To summarise, the financial viability of purchasing a UR5 to brew some coffees will depend on the owner's desired brand image for their coffee shop. If the owner desired to use the UR5 as a marketing tool to give unique experience to customers being served by a robot, buying a UR5 robot would be financially viable.

7 Takeaways from Assignment

In this project, several lessons and skills were obtained. The main skills learnt from this assignment was utilising frame assignments and homogenous transforms. A reoccurring method in this assignment was the group setting arbitrary frames for the TCP to orient the end effector in a position the group desired. Utilising arbitrary frames simplified the movement calculations.

The group also learnt how to use the homogenous transform effectively to describe frames, mapping and as an operator. The group used homogenous transforms to reach the desired end-effector positions to perform the tasks desired. Another key use of the homogenous transform in this project was the compound transformations to determine the unknown transforms.

In terms of using RoboDK, a lesson that the group learnt was to use pauses after calling Rodney's functions. Without using a pause when calling Rodney's function, weird behaviour started to occur. An example of this weird behaviour is when the portafilter tool randomly detached after reaching the end of Rodney's program and then continued running the program.

Overall, the main takeaway from this assignment is that the group learnt how to program a miniaturised industrial robot. Utilising frame assignments and homogenous transforms, the group can program the robot to complete tasks. However, this assignment also proved the inaccuracy of the simulation. The simulation of the environment inside RoboDK has major offsets compared to the real-life UR5 robot.

8 Recommendations

In this project, the group's approach to the assignment could have drastically improved. The first improvement the group could have made was the approach to the laboratory times. At the early stages of the project, the group did not make most of their valuable time during their allocated time slots as they were debugging RoboDK. For future students, the group would recommend ensuring that their program is running before their allocated time slots to have more time testing the robot. Also, the group recommend using the pause function whenever calling Rodney's functions to avoid unexplained behaviour of the robot.

The group could have also run the robot as much as possible. In this project, the group was hesitant to run individual tasks as they were too focused on running Task A-J to make a coffee. For future students, the group would recommend calibrating each task one by one and combining all tasks at the end rather than being deeply focused on thinking about running the whole program.

Additionally, the group could have started working with simple tasks first rather than working on the hard tasks. In this project, the group struggled with calibrating the portafilter tool in which several laboratory times were wasted. If the group focused on the simpler task such as pushing buttons, scraping and tamping, the group would not have to calibrate these simple tasks last minute.

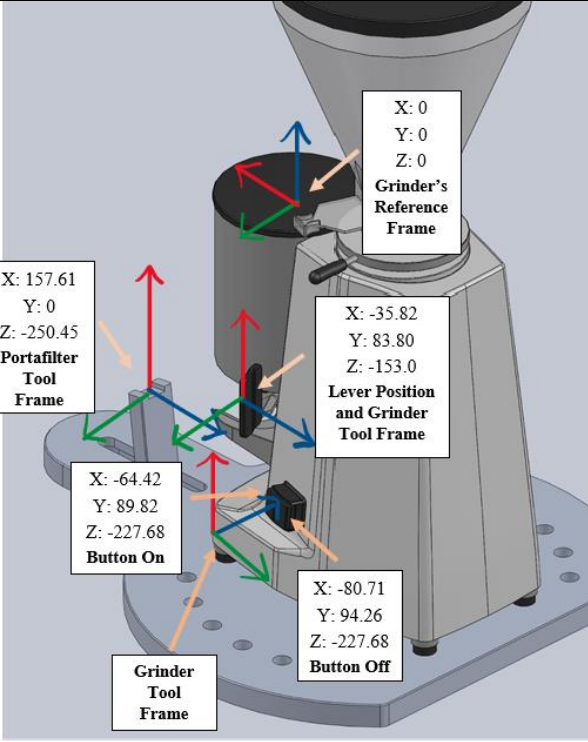
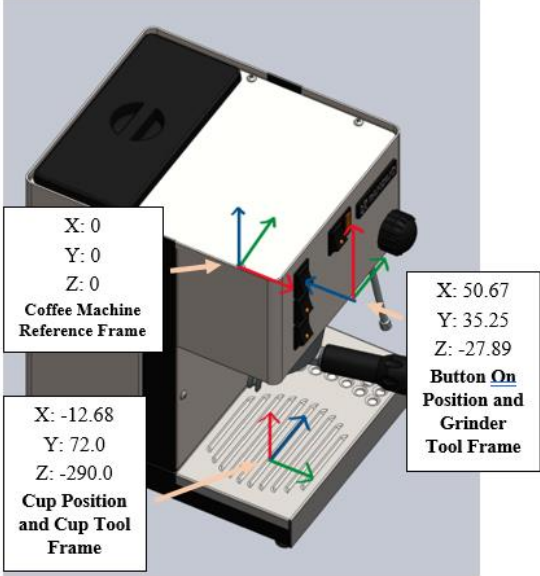
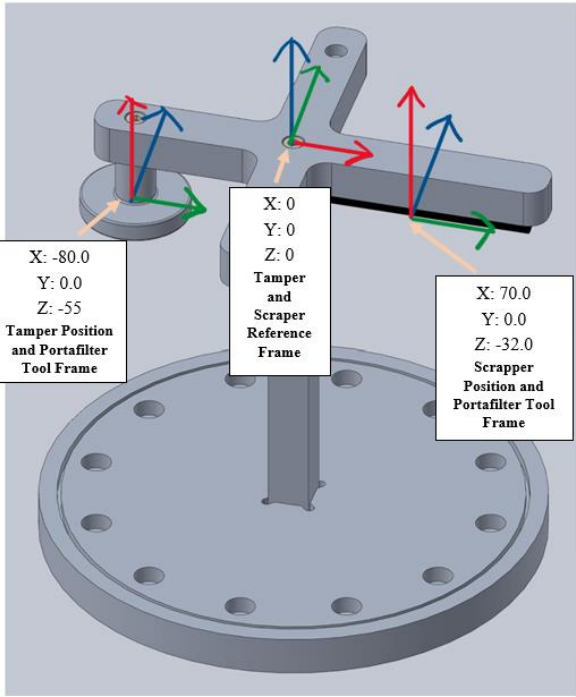
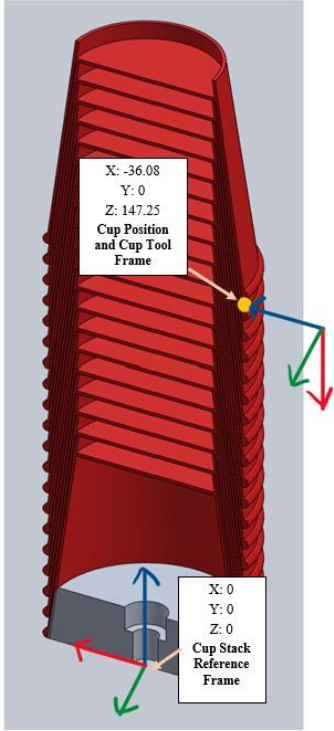
For this assignment improvement, the group recommends that the 2020 apparatus should be reviewed. The 2020 apparatus tends to have a larger offset compared to the 2019 apparatus after comparing with other groups. This is not a disadvantage for groups who are working in 2020 apparatus as they still need to calibrate, but it is something to point out.

An extended recommendation the group would make for future students is to try implementing inverse kinematics in this project. Inverse kinematics would be ideal for path planning and control as the desired end-effector configuration is known. With RoboDK, the inverse kinematics is already provided showing multiple solutions for possible joint configurations but implementing this in a python script would exercise what was taught in lectures.

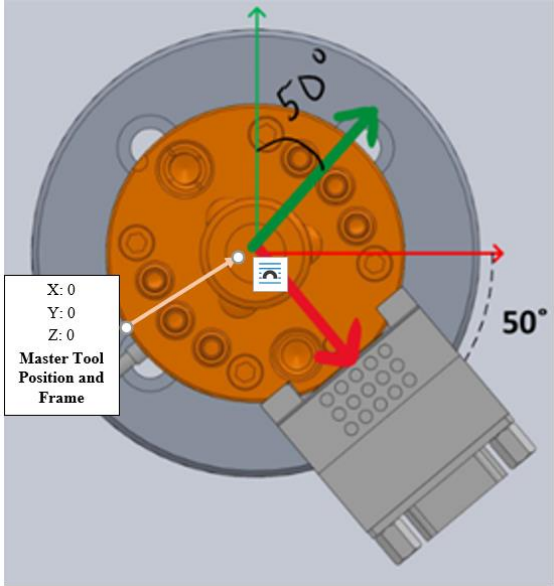
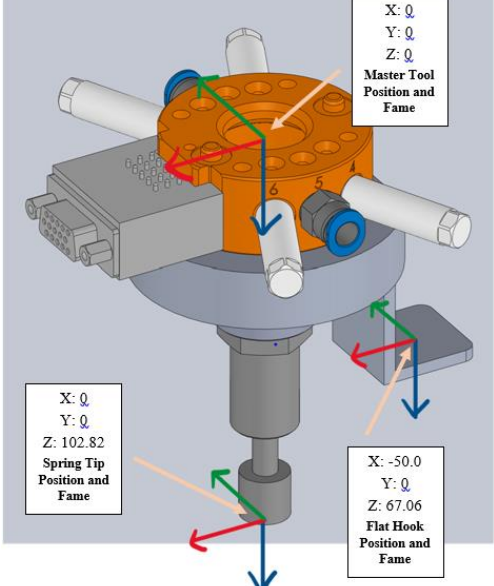
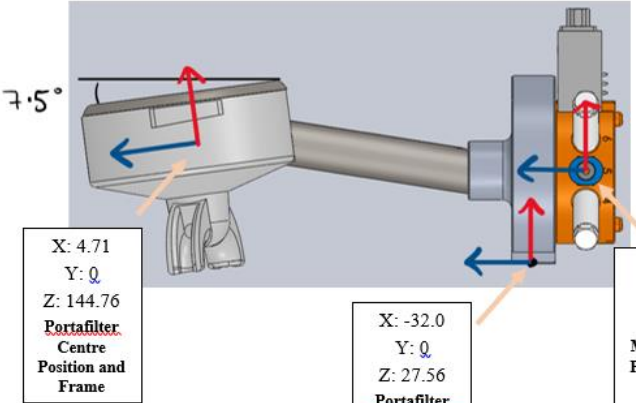
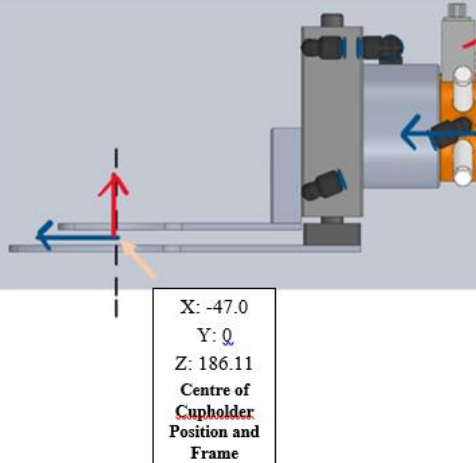
In summary from the group's experience, the main recommendation is to calibrate and test the robot as much as possible. The group also recommend completing the easiest tasks first and then work on the harder ones to reduce wasted time. Lastly, the group recommends implementing inverse kinematics without using RoboDK to exercise what is taught in lectures as DH convention for the UR5 robot is provided in Learn.

9. Appendix

A. Frame Assignments

Grinder	Coffee Machine
 <p>Diagram illustrating the coordinate frames for the Grinder components:</p> <ul style="list-style-type: none">Grinder's Reference Frame: X: 0, Y: 0, Z: 0Portafilter Tool Frame: X: 157.61, Y: 0, Z: -250.45Lever Position and Grinder Tool Frame: X: -35.82, Y: 83.80, Z: -153.0Button On: X: -64.42, Y: 89.82, Z: -227.68Grinder Tool Frame: X: -80.71, Y: 94.26, Z: -227.68Button Off: X: -80.71, Y: 94.26, Z: -227.68	 <p>Diagram illustrating the coordinate frames for the Coffee Machine components:</p> <ul style="list-style-type: none">Coffee Machine Reference Frame: X: 0, Y: 0, Z: 0Button On Position and Grinder Tool Frame: X: 50.67, Y: 35.25, Z: -27.89Cup Position and Cup Tool Frame: X: -12.68, Y: 72.0, Z: -290.0
Tamper and Scraper Stand	Cup Stack
 <p>Diagram illustrating the coordinate frames for the Tamper and Scraper Stand components:</p> <ul style="list-style-type: none">Tamper and Scraper Reference Frame: X: 0, Y: 0, Z: 0Tamper Position and Portafilter Tool Frame: X: -80.0, Y: 0.0, Z: -55Scraper Position and Portafilter Tool Frame: X: 70.0, Y: 0.0, Z: -32.0	 <p>Diagram illustrating the coordinate frames for the Cup Stack components:</p> <ul style="list-style-type: none">Cup Position and Cup Tool Frame: X: -36.08, Y: 0, Z: 147.25Cup Stack Reference Frame: X: 0, Y: 0, Z: 0

B. Tool Frame Assignments

Master Tool	Grinder Tool
 <p>X: 0 Y: 0 Z: 0 Master Tool Position and Frame</p>	 <p>X: Q Y: Q Z: 102.82 Spring Tip Position and Fame</p> <p>X: -50.0 Y: Q Z: 67.06 Flat Hook Position and Fame</p>
Portafilter Tool	
 <p>X: 4.71 Y: Q Z: 144.76 Portafilter Centre Position and Frame</p> <p>X: -32.0 Y: Q Z: 27.56 Portafilter Edge Position and Frame</p> <p>X: Q Y: Q Z: Q Master Tool Position and Frame</p>	
Cup Tool	
 <p>X: -47.0 Y: Q Z: 186.11 Centre of Cupholder Position and Frame</p> <p>X: Q Y: Q Z: Q Master Tool Position and Frame</p>	

C. Homogenous Transforms Calculations

C.1 Grinder Tool

Calculate Rotation Matrix:

$${}^{Master Tool}R_{Grinder Tool Flat Hook} = I_{3 \times 3}$$

$${}^{Master Tool}R_{Grinder Tool Spring Tip} = I_{3 \times 3}$$

Calculate Homogenous Transform:

$${}^{Master Tool}_{Grinder Tool Flat Hook} T = \begin{bmatrix} {}^{Master Tool}R_{Grinder Tool Flat Hook(3 \times 3)} & \begin{matrix} -50.0 \\ 0 \\ 67.06 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{Master Tool}_{Grinder Tool Spring Tip} T = \begin{bmatrix} {}^{Master Tool}R_{Grinder Tool Spring Tip(3 \times 3)} & \begin{matrix} 0 \\ 0 \\ 102.82 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

C.2 Cup Tool

Calculate Rotation Matrix:

$${}^{Master Tool}R_{Cup Tool} = I_{3 \times 3}$$

Calculate Homogenous Transform:

$${}^{Master Tool}_{Cup Tool} T = \begin{bmatrix} {}^{Master Tool}R_{Cup Tool(3 \times 3)} & \begin{matrix} -47.0 \\ 0 \\ 186.11 \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

D. Source Code

```
#
#
# ENMT482 Manipulators Assignment
# Authors: Jaime Sequeira and Zoren Dela Cruz
#
#

"""
-----
-----

    This python script will control a UR5 to make a cup of coffee using the
    2020 apparatus
-----
-----

"""

# Import Tasks and python scripts
from Tasks import TaskA, TaskB, TaskC, TaskD, TaskE, TaskF, TaskG, TaskH,
TaskI, TaskJ
from Scripts.ReturnHome import goHome

# Create an array of all the tasks
taskArray = [TaskA, TaskB, TaskC, TaskD, TaskE, TaskF, TaskG, TaskH, TaskI,
TaskJ]
"""
-----
-----

    And that is all there is to it. Call the main function to start making
    a fresh
    brew. xoxo
-----
-----

"""

def main ():

    # Send the robot to the starting position
    goHome()

    # Iterate through the task array and call the runTask function
    for task in taskArray:
        task.runTask()

# Run the python script
main()
```

```

"""
-----
-----

    This class is used to describe each item of equipment on the table.
    The class instances will define each equipment's global pose and
    rotation matrix.
    -----
    -----
"""

class Equipment:

    def __init__(self, name, HTransform):

        self.name = name                # Define the machine/item name just
in case it's required later on
        self.HTransform = HTransform    # Define the homogeneous transform
for the machine/item

"""
-----
-----

    This script will create a homogeneous transform.
    Parameters:
        Rotation matrix (3x3 matrix)
        Global pose vector (4x1 column vector)
    Appends the global pose vector to the rotation matrix and adds a [0, 0,
0, 1] row
    to create the Homogeneous transform
    -----
    -----
"""

def convertToHT(rotationMatrix, globalPoseVector):

    bottomRow = np.array([0, 0, 0, 1])

    HT = np.insert(rotationMatrix, 3, globalPoseVector, axis=1)
    HT = np.vstack((HT, bottomRow))

    return(HT)

"""
-----
-----

    This script will convert a global pose into the rotation matrix for an
    item on the table.

    Parameters:
        - Position vector 1 (position of the reference frame in
global coordinates (3x1 vector))
        - Position vector 2 (co-ordinate of point provided on the

```



```

machine)
-----
-----

"""

def convertToRM(positionVector1, positionVector2, angleOffset, operator):

    # Calculate the angle difference
    dx = abs(positionVector2[0] - positionVector1[0])
    dy = abs(positionVector2[1] - positionVector1[1])

    angleOffset = angleOffset * (pi/180)

    if (operator == '+'):
        theta = angleOffset + np.arctan(dy/dx)

    elif (operator == '-'):
        theta = angleOffset - np.arctan(dy/dx)

    RM = np.array([
        [cos(theta), -sin(theta), 0],
        [sin(theta), cos(theta), 0],
        [0, 0, 1]
    ])

    return RM

"""

-----
-----

    This script will allow the programmer to call any of Rodney's
    functions.

    NOTE The pause after calling Rodney's function is vital. If you remove
    it, bad things will
        happen.
    -----
    -----

"""

def attachGrinderTool():
    # Call the pre-defined tool attachment sequence
    RDK.RunProgram("Grinder Tool Attach (Stand)", True)      # Second
    parameter set to true to make the call blocking
    robodk.pause(0.1)

def detachGrinderTool():
    # Call the pre-defined tool detachment sequence
    RDK.RunProgram("Grinder Tool Detach (Stand)", True)      # Second
    parameter set to true to make the call blocking
    robodk.pause(0.1)

def attachCupTool():
    RDK.RunProgram("Cup Tool Attach (Stand)", True)
    robodk.pause(0.1)

```

```

def detachCupTool():
    RDK.RunProgram("Cup Tool Detach (Stand)", True)
    robodk.pause(0.1)

def openCupTool():
    RDK.RunProgram("Cup Tool Open", True)
    robodk.pause(0.1)

def closeCupTool():
    RDK.RunProgram("Cup Tool Close", True)
    robodk.pause(0.1)

def attachPortafilterToolStand():
    RDK.RunProgram("Portafilter Tool Attach (Stand)", True)
    robodk.pause(0.1)

def detachPortafilterToolStand():
    RDK.RunProgram("Portafilter Tool Detach (Stand)", True)
    robodk.pause(0.1)

def attachPortafilterToolGrinder():
    RDK.RunProgram("Portafilter Tool Attach (Grinder)", True)
    robodk.pause(0.1)

def detachPortafilterToolGrinder():
    RDK.RunProgram("Portafilter Tool Detach (Grinder)", True)
    robodk.pause(0.1)

def detachPortafilterToolSilvia():
    # You shouldn't need to call this since the TA will put the PF tool
    into the machine
    RDK.RunProgram("Portafilter Tool Detach (Silvia)", True)
    robodk.pause(0.1)

"""

-----
-----

    Define the global poses and rotation matrices (RM) of the machines
-----
-----

"""

grinderMachineGlobalPosition = [484.51, -426.60, 318.38]
grinderPoint2Vector = [369.74, -320.06, 67.83]
grinderMachineRM = convertToRM(grinderMachineGlobalPosition,
grinderPoint2Vector, 180.0, '-')
grinderMachineHT = convertToHT(grinderMachineRM,
grinderMachineGlobalPosition)

coffeeMachineGlobalPosition = [-359.90, -387.38, 341.24]
coffeeMachinePoint1Vector = [-359.90, -387.38, 341.24]
coffeeMachinePoint2Vector = [-573.54, -443.66, 341.24]
coffeeMachineRM = convertToRM(coffeeMachinePoint1Vector,
coffeeMachinePoint2Vector, 90.0, '+')
coffeeMachineHT = convertToHT(coffeeMachineRM, coffeeMachineGlobalPosition)

tamperGlobalPosition = [598.10, 4.31, 212.58]

```

```

tamperPoint2Vector = [557.50, 73.18, 157.58]
tamperRM = convertToRM(tamperGlobalPosition, tamperPoint2Vector, 0.0, '-')
tamperHT = convertToHT(tamperRM, tamperGlobalPosition)

# The cup stack doesn't require calculating a rotation matrix, you can just
eyeball it
cupStackGlobalPosition = [1.92, -595.89, -20.0]
cupStackRM = np.array([
    [-1.0, 0.0, 0.0],
    [0.0, -1.0, 0.0],
    [0.0, 0.0, 1.0]
])

cupStackHT = convertToHT(cupStackRM, cupStackGlobalPosition)

"""
-----
-----

    Create an instance of each piece of equipment used in the process
-----
-----

"""

GrinderMachine = Equipment('grinderMachine', grinderMachineHT)
CoffeeMachine = Equipment('coffeeMachine', coffeeMachineHT)
Tamper = Equipment('tamper', tamperHT)
CupStack = Equipment('cupStack', cupStackHT)

"""
-----
-----

    This script will execute Task A: Place portafilter tool under the
grinder dosing head.
-----
-----

"""

"""
-----
-----

    Define the parameters for the tools and machine
-----
-----

"""

masterToolTheta = (50) * (pi/180)
masterToolRM = np.array([
    [cos(masterToolTheta), sin(masterToolTheta),
0.0],
    [-sin(masterToolTheta), cos(masterToolTheta),

```

```

0.0],
                                [0.0, 0.0, 1.0]
    ])

grinderMachineStandTheta = 7.5 * (pi/180)
grinderMachinePortafilterStandRM = np.array([

[sin(grinderMachineStandTheta), 0.0, -cos(grinderMachineStandTheta)],
                                [0.0, 1.0, 0.0],

[cos(grinderMachineStandTheta), 0.0, sin(grinderMachineStandTheta)]
    ])

portafilterToolTheta = -7.5 * (pi/180)
portaFilterToolForInsertRM = np.array([
                                [cos(portafilterToolTheta), 0.0, -
sin(portafilterToolTheta)],
                                [0.0, 1.0, 0.0],
                                [sin(portafilterToolTheta), 0.0,
cos(portafilterToolTheta)]
    ])

portafilterToolRM = masterToolRM
portaFilterToolForInsertRM = masterToolRM.dot(portaFilterToolForInsertRM)

"""

-----

Define the HTs for the required positions for this task
-----

"""

portafilterToolLocal = np.array([-32.0, 0.0, 27.56])
portafilterToolHT = convertToHT(portafilterToolRM, portafilterToolLocal)

portafilterToolBeforeInsertHT = convertToHT(portaFilterToolForInsertRM,
portafilterToolLocal)

grinderMachinePortaFilterStandLocal = np.array([157.61, 0.0, -250.45])
grinderMachinePortaFilterStandLocalOffset =
grinderMachinePortaFilterStandLocal + [2.0, -24.0, 10.0]
grinderMachinePortafilterStandHT =
convertToHT(grinderMachinePortafilterStandRM,
grinderMachinePortaFilterStandLocalOffset)

filterAboveStandWaypoint = grinderMachinePortaFilterStandLocalOffset +
[30.0, 0.0, 10.0]
filterAboveStandWaypointHT = convertToHT(grinderMachinePortafilterStandRM,
filterAboveStandWaypoint)

"""

-----

```

```

    Define the robot poses for the HTs defined
    -----
    """

filterAboveStandPose =
np.array((GrinderMachine.HTransform.dot(filterAboveStandWaypointHT)).dot(np
.linalg.inv(portafilterToolBeforeInsertHT)))
filterAboveStandPose = robodk.Mat(filterAboveStandPose.tolist())

filterInStandPose =
np.array((GrinderMachine.HTransform.dot(grinderMachinePortafilterStandHT)).
dot(np.linalg.inv(portafilterToolHT)))
filterInStandPose = robodk.Mat(filterInStandPose.tolist())

"""

    Set some waypoints to avoid obstacles and achieve the correct link
    configuration
    -----
    """

moveFromToolStandWaypoint = [-123.309562, -47.728666, -94.411161, -
136.914621, 82.872853, 25.618375]
gridnerWaypoint = [-67.551267, -83.466840, -137.024044, -128.678660, -
94.772205, 137.471476]
correctOrientationBeforeInsert = [-14.104606, -86.256265, -144.419558, -
120.782978, -61.503068, 135.901442]
correctOrientationAfterInsert = [-5.769209, -90.794977, -156.428809, -
103.403123, -53.268753, 134.362027]

def moveToGrinder():
    robot.MoveJ(moveFromToolStandWaypoint, blocking=True)
    robot.MoveJ(gridnerWaypoint, blocking=True)
    robot.MoveJ(correctOrientationBeforeInsert, blocking=True)
    robot.MoveL(filterAboveStandPose, blocking=True)
    robot.MoveL(filterInStandPose, blocking=True)

def moveAwayFromGrinder():
    robot.MoveL(correctOrientationAfterInsert, blocking=True)

def runTask():
    RodneyFunctions.attachPortafilterToolStand()
    moveToGrinder()
    RodneyFunctions.detachPortafilterToolGrinder()
    moveAwayFromGrinder()
    goHome()

"""
    -----
    -----

```

```
    This script will execute Task B: Use grinder tool to turn the grinder
    on, wait 3s, turn the
    grinder off.
```

```
-----
"""
```

```
"""
```

```
-----
Define the global poses and rotation matrices (RM) of the tools and
machine
-----
```

```
"""
```

```
pushyBitLocalPosition = [0.0, 0.0, 102.82]
grinderToolTheta = 50 * (pi/180)
```

```
grinderToolRM = np.array([
    [cos(grinderToolTheta), -sin(grinderToolTheta),
0],
    [sin(grinderToolTheta), cos(grinderToolTheta),
0],
    [0, 0, 1]
])
```

```
grinderPowerButtonRM = np.array([
    [1.0, 0.0, 0.0],
    [0.0, 0.0, -1.0],
    [0.0, 1.0, 0.0]
])
```

```
grinderPowerButtonOffLocal = np.array([-80.71, 94.26, -227.68])
grinderPowerButtonOnLocal = np.array([-64.42, 89.82, -227.68])
```

```
grinderPowerButtonOffLocalOffset = grinderPowerButtonOffLocal + [-6.0, 5.0,
0.0]
grinderPowerButtonOnLocalOffset = grinderPowerButtonOnLocal + [0.0, 0.0,
0.0]
```

```
grinderPowerButtonWaypointLocal = grinderPowerButtonOnLocalOffset + ([0.0,
30.0, 0.0])
```

```
"""
```

```
-----
Define the HTs for the positions and orientations required
-----
```

```

"""

grinderPowerButtonOnHT = convertToHT(grinderPowerButtonRM,
grinderPowerButtonOnLocalOffset)
grinderPowerButtonOffHT = convertToHT(grinderPowerButtonRM,
grinderPowerButtonOffLocalOffset)
grinderPowerButtonWaypointLocalHT = convertToHT(grinderPowerButtonRM,
grinderPowerButtonWaypointLocal)
grinderToolPushyBitHT = convertToHT(grinderToolRM, pushyBitLocalPosition)

"""

-----
Find the poses by multiplying the HTs
-----

"""

turnOnPose =
np.array((GrinderMachine.HTransform.dot(grinderPowerButtonOnHT)).dot(np.linalg.inv(grinderToolPushyBitHT)))
turnOnPose = robodk.Mat(turnOnPose.tolist())

turnOffPose =
np.array((GrinderMachine.HTransform.dot(grinderPowerButtonOffHT)).dot(np.linalg.inv(grinderToolPushyBitHT)))
turnOffPose = robodk.Mat(turnOffPose.tolist())

waypointPose =
np.array((GrinderMachine.HTransform.dot(grinderPowerButtonWaypointLocalHT)).dot(np.linalg.inv(grinderToolPushyBitHT)))
waypointPose = robodk.Mat(waypointPose.tolist())

"""

-----
Set waypoints to avoid obstacles and achieve the correct link
configuration
-----

"""

grinderMachineButtonsWaypoint = [94.050768, -39.556615, 43.283352, -
3.726737, -43.078946, -50.000000]
correctOrientation = [102.386192, -20.123820, 27.601568, -7.477748, -
34.743522, -50.000000]

def turnOnGrinder():
    # Move the TCP to turn on the machine
    robot.MoveJ(grinderMachineButtonsWaypoint, blocking=True)
    robot.MoveL(correctOrientation, blocking=True)
    robot.MoveL(waypointPose, blocking=True)
    robot.MoveL(turnOnPose, blocking=True)

def turnOffGrinder():
    robot.MoveJ(waypointPose, blocking=True)
    robot.MoveL(turnOffPose, blocking=True)
    robot.MoveJ(waypointPose, blocking=True)

def runTask():
    RodneysFunctions.attachGrinderTool()
    turnOnGrinder()

```

```

turnOffGrinder()

"""
-----
This script will execute Task C: Use grinder tool to pull the grinder
dosing lever to deposit
ground coffee in the portafilter tool.
-----
"""

"""
-----
Define the parameters for the machine and tools
-----
"""
masterToolTheta = (50) * (pi/180)
masterToolRM = np.array([
    [cos(masterToolTheta), sin(masterToolTheta),
0.0],
    [-sin(masterToolTheta), cos(masterToolTheta),
0.0],
    [0.0, 0.0, 1.0]
])

grinderToolPullyBitRM = masterToolRM
grinderToolPullyBitPointLocal = np.array([-50.0, 0.0, 67.06])
grinderToolPullyBitHT = convertToHT(grinderToolPullyBitRM,
grinderToolPullyBitPointLocal)

grinderMachineLeverRM = np.array([
    [0.0, 0.0, -1.0],
    [1.0, 0.0, 0.0],
    [0.0, -1.0, 0.0]
])

leverEndTheta = 30 * (pi/180)
leverEndRM = np.array([
    [cos(leverEndTheta), 0.0, -sin(leverEndTheta)],
    [sin(leverEndTheta), 0.0, cos(leverEndTheta)],
    [0.0, -1.0, 0.0]
])

"""
-----
Define the HTs for the positions and orientations required
-----
"""
grinderMachineLeverStartPointLocal = np.array([-35.82, 115.82, -153.0])
grinderMachineLeverStartPointLocalOffset =
grinderMachineLeverStartPointLocal + [-3.0, 5.0, 65.0]
grinderMachineLeverStartHT = convertToHT(grinderMachineLeverRM,
grinderMachineLeverStartPointLocalOffset)

leverStartWaypointLocal = grinderMachineLeverStartPointLocalOffset + [0.0,

```



```

40.0, 0.0]
leverStartWaypointHT = convertToHT(grinderMachineLeverRM,
leverStartWaypointLocal)

leverMidpointLocal = grinderMachineLeverStartPointLocalOffset + [50.0, 0.0,
0.0]
leverMidpointHT = convertToHT(grinderMachineLeverRM, leverMidpointLocal)

grinderMachineLeverEndPointLocal = grinderMachineLeverStartPointLocalOffset
+ [160.0, -55.0, 0.0]
grindeMachineLeverEndHT = convertToHT(leverEndRM,
grinderMachineLeverEndPointLocal)

"""
-----
Define the poses required to achieve the correct orientations and
positions
-----
"""
leverStartPointPose =
np.array((GrinderMachine.HTransform.dot(grinderMachineLeverStartHT)).dot(np
.linalg.inv(grinderToolPullyBitHT)))
leverStartPointPose = robodk.Mat(leverStartPointPose.tolist())

leverMidpointPose =
np.array((GrinderMachine.HTransform.dot(leverMidpointHT)).dot(np.linalg.inv
(grinderToolPullyBitHT)))
leverMidpointPose = robodk.Mat(leverMidpointPose.tolist())

leverEndPointPose =
np.array((GrinderMachine.HTransform.dot(grindeMachineLeverEndHT)).dot(np.li
nalg.inv(grinderToolPullyBitHT)))
leverEndPointPose = robodk.Mat(leverEndPointPose.tolist())

leverStartWaypointPose =
np.array((GrinderMachine.HTransform.dot(leverStartWaypointHT)).dot(np.linal
g.inv(grinderToolPullyBitHT)))
leverStartWaypointPose = robodk.Mat(leverStartWaypointPose.tolist())

"""
-----
Set waypoints to avoid obstacles
-----
"""
moveFromButtonToLeverPosition = [104.956371, -65.712969, 98.213957, -
32.500988, 57.826657, -130.000000]
returningToolWaypoint = [104.582873, -97.216516, 124.564230, -142.028342, -
74.717391, 43.619592]
orientationBeforeRemovingTool = [-130.329671, -80.645595, -75.771770, -
113.734052, 89.970275, -156.060446]

def moveToLever():
    robot.MoveJ(moveFromButtonToLeverPosition, blocking=True)
    robot.MoveL(leverStartWaypointPose, blocking=True)
    robot.MoveL(leverStartPointPose, blocking=True)

```

```

def useDoser():
    robot.MoveL(leverMidpointPose, blocking=True)
    robot.MoveL(leverEndPointPose, blocking=True)
    robot.MoveL(leverStartWaypointPose, blocking=True)
    robot.MoveL(leverStartPointPose, blocking=True)

def moveFromDoser():
    # Send back to the waypoint
    robot.MoveL(leverStartWaypointPose, blocking=True)
    robot.MoveL(returningToolWaypoint, blocking=True)
    robot.MoveJ(orientationBeforeRemovingTool, blocking=True)

def runTask():

    moveToLever()

    for i in range(0, 3):
        useDoser() # Pull the doser lever 3 times

    moveFromDoser()

    # Send the tool back for the next task
    RodneyFunctions.detachGrinderTool()

"""
-----
This script will execute Task D: Scrape coffee from the rim of
portafilter tool
-----
"""

"""
-----
Define constraints for dealing with the portafilter in the machine
since we will have to pick
up the portafilter tool in the grinder
-----
"""

masterToolTheta = (50) * (pi/180)
masterToolRM = np.array([
    [cos(masterToolTheta), sin(masterToolTheta),
0.0],
    [-sin(masterToolTheta), cos(masterToolTheta),
0.0],
    [0.0, 0.0, 1.0]
])

grinderMachineStandTheta = 7.5 * (pi/180)
grinderMachinePortafilterStandRM = np.array([
    [sin(grinderMachineStandTheta), 0.0, -cos(grinderMachineStandTheta)],
    [0.0, 1.0, 0.0],
    [cos(grinderMachineStandTheta), 0.0, sin(grinderMachineStandTheta)]
])

```

```

portafilterToolRM = masterToolRM
portafilterToolLocal = np.array([-32.0, 0.0, 27.56])
portafilterToolHT = convertToHT(portafilterToolRM, portafilterToolLocal)

grinderMachinePortaFilterStandLocal = np.array([157.61, 0.0, -250.45])
grinderMachinePortaFilterStandLocalOffset =
grinderMachinePortaFilterStandLocal + [0.0, -25.0, 8.0]
grinderMachinePortafilterStandHT =
convertToHT(grinderMachinePortafilterStandRM,
grinderMachinePortaFilterStandLocalOffset)

grinderMachinePortaFilterStandWaypointLocal =
grinderMachinePortaFilterStandLocalOffset + [20.0, 0.0, -2.0]
grinderMachinePortaFilterStandWaypointHT =
convertToHT(grinderMachinePortafilterStandRM,
grinderMachinePortaFilterStandWaypointLocal)

filterInStandWaypointPose =
np.array((GrinderMachine.HTransform.dot(grinderMachinePortaFilterStandWaypo
intHT)).dot(np.linalg.inv(portafilterToolHT)))
filterInStandWaypointPose = robodk.Mat(filterInStandWaypointPose.tolist())

filterReturnWaypointLocal = grinderMachinePortaFilterStandLocalOffset +
[80.0, 0.0, 0.0]
filterReturnWaypointHT = convertToHT(grinderMachinePortafilterStandRM,
filterReturnWaypointLocal)
filterReturnWaypointPose =
np.array((GrinderMachine.HTransform.dot(filterReturnWaypointHT)).dot(np.lin
alg.inv(portafilterToolHT)))
filterReturnWaypointPose = robodk.Mat(filterReturnWaypointPose.tolist())

"""
-----
Define constraints for dealing with the portafilter in the environment
since we have to move the
portafilter tool parallel to the table to avoid spilling beans
-----
"""
thetaPortafilterCup = 12.5 * (pi/180)
portafilterCupRM = np.array([
    [cos(thetaPortafilterCup), 0.0, -
sin(thetaPortafilterCup)],
    [0.0, 1.0, 0.0],
    [sin(thetaPortafilterCup), 0.0,
cos(thetaPortafilterCup)]
])

portafilterCupRM = portafilterCupRM.dot(masterToolRM)

portafilterCupPointLocal = np.array([4.71, 0.0, 144.76])
portafilterCupHT = convertToHT(portafilterCupRM, portafilterCupPointLocal)

scraperRM = np.array([
    [0.0, 1.0, 0.0],
    [0.0, 0.0, 1.0],
    [1.0, 0.0, 0.0]
])

```

```

"""
-----
    Define the HTs for the positions and orientations required
-----
"""
scraperPointLocal = np.array([70.0, 0.0, -32.0])
scraperPointLocalOffset = scraperPointLocal + [0.0, 0.0, 2.0]
scraperHT = convertToHT(scraperRM, scraperPointLocalOffset)

scraperStartPoint = scraperPointLocalOffset + [0.0, -80.0, -20.0]
scraperEndPoint = scraperPointLocalOffset + [0.0, 20.0, -20.0]

scraperStartPointHT = convertToHT(scraperRM, scraperStartPoint)
scraperEndPointHT = convertToHT(scraperRM, scraperEndPoint)

"""
-----
    Define the poses required to achieve the correct orientation and
    position
-----
"""
scraperStartPointPose =
np.array((Tamper.HTransform.dot(scraperStartPointHT)).dot(np.linalg.inv(portafilterCupHT)))
scraperStartPointPose = robodk.Mat(scraperStartPointPose.tolist())

scraperEndPointPose =
np.array(Tamper.HTransform.dot(scraperEndPointHT).dot(np.linalg.inv(portafilterCupHT)))
scraperEndPointPose = robodk.Mat(scraperEndPointPose.tolist())

"""
-----
    Define some waypoints to avoid robot collisions
-----
"""
grinderWaypoint = [-50.308670, -69.876857, -141.733990, -140.897530, -
95.747565, 140.788317]
correctOrientationToAttachPFTool = [-17.964519, -100.772848, -140.065492, -
110.807402, -62.880571, 136.220429]

def moveToGrinderTool():
    robot.MoveJ(grinderWaypoint, blocking=True)
    robot.MoveJ(correctOrientationToAttachPFTool, blocking=True)
    robot.MoveL(filterInStandWaypointPose, blocking=True)

def moveAwayFromGrinderTool():
    robot.MoveL(filterReturnWaypointPose, blocking=True)

def scrapeCoffee():
    robot.MoveJ(scraperStartPointPose, blocking=True)
    robot.MoveJ(scraperEndPointPose, blocking=True)
    robot.MoveL(scraperStartPointPose, blocking=True)

```

```

def runTask ():
    moveToGrinderTool()
    RodneysFunctions.attachPortafilterToolGrinder()
    moveAwayFromGrinderTool()
    scrapeCoffee()

"""
-----
This script will execute Task E: Tamp coffee.
-----
"""

"""
-----
Define constraints for dealing with the portafilter in the environment
since we have to move the
    portafilter tool parallel to the table to avoid spilling beans
-----
"""
masterToolTheta = (50) * (pi/180)
masterToolRM = np.array([
    [cos(masterToolTheta), sin(masterToolTheta),
0.0],
    [-sin(masterToolTheta), cos(masterToolTheta),
0.0],
    [0.0, 0.0, 1.0]
])

thetaPortafilterCup = 12.5 * (pi/180)
portafilterCupRM = np.array([
    [cos(thetaPortafilterCup), 0.0, -
sin(thetaPortafilterCup)],
    [0.0, 1.0, 0.0],
    [sin(thetaPortafilterCup), 0.0,
cos(thetaPortafilterCup)]
])

portafilterCupRM = portafilterCupRM.dot(masterToolRM)

portafilterCupPointLocal = np.array([4.71, 0.0, 144.76])
portafilterCupHT = convertToHT(portafilterCupRM, portafilterCupPointLocal)

tamperToolRM = np.array([
    [0.0, 1.0, 0.0],
    [0.0, 0.0, 1.0],
    [1.0, 0.0, 0.0]
])

"""
-----
Define the HTs for the positions and orientations required
-----
"""

```

```

"""
tamperToolPointLocal = np.array([-80.0, 0.0, -55.0])
tamperToolPointLocalOffset = tamperToolPointLocal + [15.0, 0.0, 0.0]
tamperToolHT = convertToHT(tamperToolRM, tamperToolPointLocalOffset)

tamperStartPoint = tamperToolPointLocalOffset + [0.0, 0.0, -50.0]
tamperEndPoint = tamperToolPointLocalOffset + [0.0, 0.0, -15.0]

tamperStartPointHT = convertToHT(tamperToolRM, tamperStartPoint)
tamperEndPointHT = convertToHT(tamperToolRM, tamperEndPoint)

"""
-----
Define the poses to achieve the correct positions and orientations
required
-----
"""
tamperStartPointPose =
np.array((Tamper.HTransform.dot(tamperStartPointHT)).dot(np.linalg.inv(porta
filterCupHT)))
tamperStartPointPose = robodk.Mat(tamperStartPointPose.tolist())

tamperEndPointPose =
np.array(Tamper.HTransform.dot(tamperEndPointHT).dot(np.linalg.inv(portafil
terCupHT)))
tamperEndPointPose = robodk.Mat(tamperEndPointPose.tolist())

"""
-----
Define waypoints to avoid collisions
-----
"""
tamperStartOrientationWaypoint = [15.302765, -93.435674, -154.368575, -
104.160988, -95.524396, 141.453902]

def moveToTamper():
    robot.MoveL(tamperStartOrientationWaypoint, blocking=True)
    robot.MoveL(tamperStartPointPose, blocking=True)

def tampCoffee():
    robot.MoveL(tamperEndPointPose, blocking=True)
    robot.MoveL(tamperStartPointPose, blocking=True)

def runTask():
    moveToTamper()
    tampCoffee()

"""
-----
This script will execute Task F: Move portafilter tool to the coffee
machine (Silvia) standoff
location (TA will manually insert into coffee machine).
-----
"""
"""

```

```

-----
Define constraints for dealing with the portafilter in the environment
since we have to move the
portafilter tool parallel to the table to avoid spilling beans
-----

```

```

"""
masterToolTheta = (50) * (pi/180)
masterToolRM = np.array([
    [cos(masterToolTheta), sin(masterToolTheta),
0.0],
    [-sin(masterToolTheta), cos(masterToolTheta),
0.0],
    [0.0, 0.0, 1.0]
])

thetaPortafilterCup = 12.5 * (pi/180)
portafilterCupRM = np.array([
    [cos(thetaPortafilterCup), 0.0, -
sin(thetaPortafilterCup)],
    [0.0, 1.0, 0.0],
    [sin(thetaPortafilterCup), 0.0,
cos(thetaPortafilterCup)]
])

portafilterCupRM = portafilterCupRM.dot(masterToolRM)

portafilterCupPointLocal = np.array([4.71, 0.0, 144.76])
portafilterCupHT = convertToHT(portafilterCupRM, portafilterCupPointLocal)

"""
-----
Define constraints for the coffee machine
-----
"""
coffeeMachineGrinderAttachmentRM = np.array([
    [0.0, 0.0, -1.0],
    [0.0, 1.0, 0.0],
    [1.0, 0.0, 0.0]
])

coffeeMachineGrinderAttachmentLocalPoint = np.array([180.0, 50.0, -150.0])
coffeeMachineGrinderAttachmentHT =
convertToHT(coffeeMachineGrinderAttachmentRM,
coffeeMachineGrinderAttachmentLocalPoint)

insertPortafilterInMachinePose =
np.array((CoffeeMachine.HTransform.dot(coffeeMachineGrinderAttachmentHT)).d
ot(np.linalg.inv(portafilterCupHT)))
insertPortafilterInMachinePose =
robodk.Mat(insertPortafilterInMachinePose.tolist())

moveFromTammerWaypoint = [8.330000, -92.050000, -161.210000, -98.670000, -
97.480000, 141.730000]
rotationalWaypoint = [-52.520047, -98.099886, -147.685535, -106.190319, -
94.361048, 141.286735]
waypoint = [-87.473269, -89.002611, -155.813596, -107.159291, -85.481602,
140.039030]

```

```

rotationalWaypoint2 = [-137.520000, -98.100000, -147.690000, -106.190000, -
94.360000, 141.290000]

def moveToCoffeeMachine():
    robot.MoveJ(moveFromTammerWaypoint, blocking=True)
    robot.MoveL(rotationalWaypoint, blocking=True)
    robot.MoveL(waypoint, blocking=True)
    robot.MoveL(rotationalWaypoint2, blocking=True)
    robot.MoveL(insertPortafilterInMachinePose, blocking=True)

def allowTAToContinue():
    robodk.pause(5)

def runTask():
    moveToCoffeeMachine()
    allowTAToContinue()

"""
-----
This script will execute Task G: Pick up a coffee cup with cup tool
-----
"""

"""
-----
Define the parameters for the machineand tools
-----
"""
thetaMasterTool = 50 * (pi/180)
cupToolRM = np.array([
    [cos(thetaMasterTool), sin(thetaMasterTool),
0.0],
    [-sin(thetaMasterTool), cos(thetaMasterTool),
0.0],
    [0.0, 0.0, 1.0]
])

cupToolPointLocal = np.array([-47.0, 0.0, 186.11])
cupToolHT = convertToHT(cupToolRM, cupToolPointLocal)

cupStackToolRM = np.array([
    [0.0, -1.0, 0.0],
    [0.0, 0.0, 1.0],
    [-1.0, 0.0, 0.0]
])

cupStackLocalPoint = np.array([-36.08, 0.0, 147.25])
cupStackLocalPointOffset = cupStackLocalPoint + [75.0, 0.0, 0.0]

cupStackStartPoint = cupStackLocalPointOffset + [0.0, -90.0, 0.0]
cupStackEndPoint = cupStackLocalPointOffset + [0.0, 0.0, 100.0]

"""
-----
Define the HTs for the positions and orientations required

```



```

-----
"""
cupStackPickUpPointHT = convertToHT(cupStackToolRM,
cupStackLocalPointOffset)
cupStackStartPointHT = convertToHT(cupStackToolRM, cupStackStartPoint)
cupStackEndPointPointHT = convertToHT(cupStackToolRM, cupStackEndPoint)

"""
-----
Define the poses to achieve the positions and orientations required
-----
"""
collectCupPose =
np.array((CupStack.HTransform.dot(cupStackPickUpPointHT)).dot(np.linalg.inv
(cupToolHT)))
collectCupPose = robodk.Mat(collectCupPose.tolist())

cupStackStartPointPose =
np.array((CupStack.HTransform.dot(cupStackStartPointHT)).dot(np.linalg.inv(
cupToolHT)))
cupStackStartPointPose = robodk.Mat(cupStackStartPointPose.tolist())

cupStackEndPointPose =
np.array((CupStack.HTransform.dot(cupStackEndPointPointHT)).dot(np.linalg.i
nv(cupToolHT)))
cupStackEndPointPose = robodk.Mat(cupStackEndPointPose.tolist())

"""
-----
Define some joint positions to avoid collisions
-----
"""
collectingCupToolWaypoint = [-133.393021, -48.089367, -144.920566, -
76.123670, 91.408056, -158.449130]
waypoint = [37.786027, -83.856132, 154.865142, -71.009010, 126.536022, -
40.000000]
cupStackRobotOrientation = [62.956931, -90.874220, 146.387035, -55.512815,
62.956931, -40.000000]

def collectCupTool():
    robot.MoveJ(collectingCupToolWaypoint, blocking=True)
    RodneyFunctions.attachCupTool()

def collectCup():
    robot.MoveL(waypoint, blocking=True)
    robot.MoveJ(cupStackStartPointPose, blocking=True)
    robodk.pause(1)
    RodneyFunctions.openCupTool()
    robot.MoveL(collectCupPose, blocking=True)
    robodk.pause(1)
    RodneyFunctions.closeCupTool()
    robodk.pause(1)

def moveFromCupStack():
    robot.MoveL(cupStackEndPointPose, blocking=True)

```

```

def runTask ():
    collectCupTool()
    # Rodney's function does not set the tool frame back to normal
    # so here is a patch to do it for you
    masterToolFrame = RDK.Item('Master Tool')
    robot.setPoseTool(masterToolFrame.Pose())

    collectCup()
    moveFromCupStack()

"""
-----
-----
    This script will execute Task H: Place a coffee cup on the drip-tray of
the coffee machine under
    the portafilter tool.
-----
-----
"""

"""
-----
-----
    Define the cup tool parameters
-----
-----
"""
thetaMasterTool = 50 * (pi/180)
cupToolRM = np.array([
                                [cos(thetaMasterTool), sin(thetaMasterTool),
0.0],
                                [-sin(thetaMasterTool), cos(thetaMasterTool),
0.0],
                                [0.0, 0.0, 1.0]
                            ])

cupToolPointLocal = np.array([-47.0, 0.0, 186.11])
cupToolHT = convertToHT(cupToolRM, cupToolPointLocal)

"""
-----
-----
    Define the coffee machine parameters
-----
-----
"""
dripTrayRM = np.array([
                                [0.0, 1.0, 0.0],
                                [0.0, 0.0, 1.0],
                                [1.0, 0.0, 0.0]
                            ])

dripTrayPointLocal = np.array([-12.68, 72.0, -290.0])
cupInDripTrayPoint = dripTrayPointLocal + [-20.0, -10.0, 85.0]
cupInMachineReturnPoint = cupInDripTrayPoint + [80.0, 0.0, 0.0]

cupInDripTrayHT = convertToHT(dripTrayRM, cupInDripTrayPoint)

cupInDripTrayPose =
np.array((CoffeeMachine.HTransform.dot(cupInDripTrayHT)).dot(np.linalg.inv(

```

```

cupToolHT)))
cupInDripTrayPose = robodk.Mat(cupInDripTrayPose.tolist())

"""
-----
Define some joint positions to avoid collisions
-----
"""
cupOrientation = [-53.930000, -84.590000, -135.010000, -140.400000, -
1.680000, 140.000000]
insertWaypoint = [-75.827199, -85.235792, -135.797750, -138.966459, -
0.585546, 140.000000]

def placeCupOnTray():
    robot.MoveJ(cupOrientation, blocking=True)
    robot.MoveJ(insertWaypoint, blocking=True)
    robot.MoveJ(cupInDripTrayPose, blocking=True)
    robodk.pause(1)

def moveAwayFromMachine():
    robot.MoveJ(insertWaypoint, blocking=True)
    RodneyFunctions.closeCupTool()
    robodk.pause(1)

def runTask():
    placeCupOnTray()
    RodneyFunctions.openCupTool()
    pause(2)
    moveAwayFromMachine()
    RodneyFunctions.detachCupTool()

"""
-----
This script will execute Task I: Use grinder tool to turn coffee
machine on, wait 3s, and turn off
the machine.
-----
"""
"""
-----
We first need to define the tool HTs and the HTs for the items on the
machine. We can start by
defining the local positions of equipment eg buttons on the machine
-----
"""
pushyBitLocalPosition = [0.0, 0.0, 102.82]
grinderToolTheta = 50 * (pi/180)

coffeeMachineButtonCentrePointLocal = np.array([50.67, 45.25, -27.89])

coffeeMachineTopButtonOn = coffeeMachineButtonCentrePointLocal + [0.0, 0.0,
5.0]
coffeeMachineTopButtonOff = coffeeMachineButtonCentrePointLocal + [0.0,
0.0, -5.0]

```

```

coffeeMachineOnWaypoint = coffeeMachineTopButtonOn + [50.0, 0.0, 0.0]
coffeeMachineOffWaypoint = coffeeMachineTopButtonOff + [50.0, 0.0, 0.0]

coffeeMachinePowerButtonWaypoint = coffeeMachineButtonCentrePointLocal +
[20.0, -50.0, 0.0]
coffeeMachineReturnWaypoint = coffeeMachineButtonCentrePointLocal + [70.0,
-100.0, 0.0]

"""
-----
We can now define the rotation matrices
-----
"""
grinderToolRM = np.array([
    [cos(grinderToolTheta), sin(grinderToolTheta),
0.0],
    [-sin(grinderToolTheta), cos(grinderToolTheta),
0.0],
    [0.0, 0.0, 1.0]
])

coffeeMachineButtonRM = np.array([
    [0.0, 0.0, -1.0],
    [0.0, 1.0, 0.0],
    [1.0, 0.0, 0.0]
])

"""
-----
And finally, we can calculate the homogeneous transforms
-----
"""
coffeeMachinePowerButtonWaypointHT = convertToHT(coffeeMachineButtonRM,
coffeeMachinePowerButtonWaypoint)
coffeeMachineOnTopButtonHT = convertToHT(coffeeMachineButtonRM,
coffeeMachineTopButtonOn)
coffeeMachineOffTopButtonHT = convertToHT(coffeeMachineButtonRM,
coffeeMachineTopButtonOff)
coffeeMachineOnWaypointHT = convertToHT(coffeeMachineButtonRM,
coffeeMachineOnWaypoint)
coffeeMachineOffWaypointHT = convertToHT(coffeeMachineButtonRM,
coffeeMachineOffWaypoint)
coffeeMachineReturnWaypointHT = convertToHT(coffeeMachineButtonRM,
coffeeMachineReturnWaypoint)

grinderToolPushyBitHT = convertToHT(grinderToolRM, pushyBitLocalPosition)

"""
-----
We can now calculate the poses for each part to be used by the robot
-----
"""
topButtonCalibrationPose =
np.array((CoffeeMachine.HTTransform.dot(coffeeMachinePowerButtonWaypointHT))

```

```

.dot(np.linalg.inv(grinderToolPushyBitHT)))
topButtonCalibrationPose = robodk.Mat(topButtonCalibrationPose.tolist())

topButtonOnPose =
np.array((CoffeeMachine.HTransform.dot(coffeeMachineOnTopButtonHT)).dot(np.
linalg.inv(grinderToolPushyBitHT)))
topButtonOnPose = robodk.Mat(topButtonOnPose.tolist())

topButtonOffPose =
np.array((CoffeeMachine.HTransform.dot(coffeeMachineOffTopButtonHT)).dot(np.
linalg.inv(grinderToolPushyBitHT)))
topButtonOffPose = robodk.Mat(topButtonOffPose.tolist())

coffeeMachineOnWaypointPose =
np.array((CoffeeMachine.HTransform.dot(coffeeMachineOnWaypointHT)).dot(np.
linalg.inv(grinderToolPushyBitHT)))
coffeeMachineOnWaypointPose =
robodk.Mat(coffeeMachineOnWaypointPose.tolist())

coffeeMachineOffWaypointPose =
np.array((CoffeeMachine.HTransform.dot(coffeeMachineOffWaypointHT)).dot(np.
linalg.inv(grinderToolPushyBitHT)))
coffeeMachineOffWaypointPose =
robodk.Mat(coffeeMachineOffWaypointPose.tolist())

coffeeMachineReturnWaypointPose =
np.array((CoffeeMachine.HTransform.dot(coffeeMachineReturnWaypointHT)).dot(
np.linalg.inv(grinderToolPushyBitHT)))
coffeeMachineReturnWaypointPose =
robodk.Mat(coffeeMachineReturnWaypointPose.tolist())

"""
-----
Now we split the task into sub tasks/functions
-----
"""
def turnOnCoffeeMachine():
    robot.MoveL(topButtonCalibrationPose, blocking=True)    # Move to the
waypoint to avoid collisions
    robot.MoveL(coffeeMachineOnWaypointPose, blocking=True)
    robot.MoveL(topButtonOnPose, blocking=True)
    robodk.pause(2)
    robot.MoveL(coffeeMachineOnWaypointPose, blocking=True)

def turnOffCoffeeMachine():
    robot.MoveL(coffeeMachineOffWaypointPose, blocking=True)
    robot.MoveL(topButtonOffPose, blocking=True)
    robodk.pause(2)
    robot.MoveL(coffeeMachineOffWaypointPose, blocking=True)
    robot.MoveL(coffeeMachineReturnWaypointPose, blocking=True)

"""
-----
We can now create the task which is called in the main function in
../group11.... robodk file
-----
"""
def runTask ():

```

```

RodneysFunctions.attachGrinderTool()
turnOnCoffeeMachine()
robodk.pause(3)
turnOffCoffeeMachine()
robodk.pause(1)
RodneysFunctions.detachGrinderTool()

"""
-----
-----
    This script will execute Task J: Hand cup of steaming coffee to
Rodney!!
-----
-----
"""

"""
-----
-----
    Define the cup tool parameters
-----
-----
"""
thetaMasterTool = 50 * (pi/180)
cupToolRM = np.array([
                                [cos(thetaMasterTool), sin(thetaMasterTool),
0.0],
                                [-sin(thetaMasterTool), cos(thetaMasterTool),
0.0],
                                [0.0, 0.0, 1.0]
])

cupToolPointLocal = np.array([-47.0, 0.0, 186.11])
cupToolHT = convertToHT(cupToolRM, cupToolPointLocal)

"""
-----
-----
    Define the coffee machine parameters
-----
-----
"""
dripTrayRM = np.array([
                                [0.0, 1.0, 0.0],
                                [0.0, 0.0, 1.0],
                                [1.0, 0.0, 0.0]
])

dripTrayPointLocal = np.array([-12.68, 72.0, -290.0])
cupInDripTrayPoint = dripTrayPointLocal + [-20.0, -10.0, 85.0]
cupInMachineReturnPoint = cupInDripTrayPoint + [80.0, 0.0, 0.0]

"""
-----
-----
    Define the HTs for the positions and orientations required
-----
-----
"""

```

```

cupInDripTrayHT = convertToHT(dripTrayRM, cupInDripTrayPoint)

"""
-----
Define the poses to achieve the positions and orientations required
-----
"""
cupInDripTrayPose =
np.array((CoffeeMachine.HTransform.dot(cupInDripTrayHT)).dot(np.linalg.inv(
cupToolHT)))
cupInDripTrayPose = robodk.Mat(cupInDripTrayPose.tolist())

"""
-----
Define some joint positions to avoid collisions
-----
"""
cupOrientation = [-53.930000, -84.590000, -135.010000, -140.400000, -
1.680000, 140.000000]
insertWaypoint = [-75.827199, -85.235792, -135.797750, -138.966459, -
0.585546, 140.000000]
giveCupToRodneyOrientation = [-182.830000, -84.240000, -135.800000, -
138.970000, -0.590000, 140.000000]

def moveToWaypoint():
    robot.MoveJ(insertWaypoint, blocking=True)

def moveToCupInMachine():
    robot.MoveJ(cupInDripTrayPose, blocking=True)

def giveCupToRodney():
    robot.MoveJ(insertWaypoint, blocking=True)
    robot.MoveJ(giveCupToRodneyOrientation, blocking=True)

def runTask():
    RodneyFunctions.attachCupTool()
    moveToWaypoint()
    RodneyFunctions.openCupTool()
    moveToCupInMachine()
    robodk.pause(2)
    RodneyFunctions.closeCupTool()
    giveCupToRodney()

```