Robot Visuomotor Control: Simulation of UR10 Robotic Arm Manipulator Catching a Flying Object

PDE-3433 Advanced Robotics

Elena Zoretich

M00725319

Eris Chinellato

Jwaad Hussain

Task Introduction

The aim of this report is to describe the environment of the given task, as well as the method used to develop the algorithm that allows the participant to reach an efficient problem's solution.

The main objective of the project is to write a program which will allow a UR10 manipulator, placed in a simulated environment, to catch a flying object thrown within the robotic arm's workspace. The flying object will fall from a point above the robot at a certain velocity and direction, and its trajectory and landing coordinates will be unknown to the user.

For the object to be considered caught, the gap between it and the manipulator's end effector must reach a minimum threshold distance of 5cm and orientation angle of 5 degrees simultaneously, at which point the simulator will generate a new, different trajectory for the falling object.

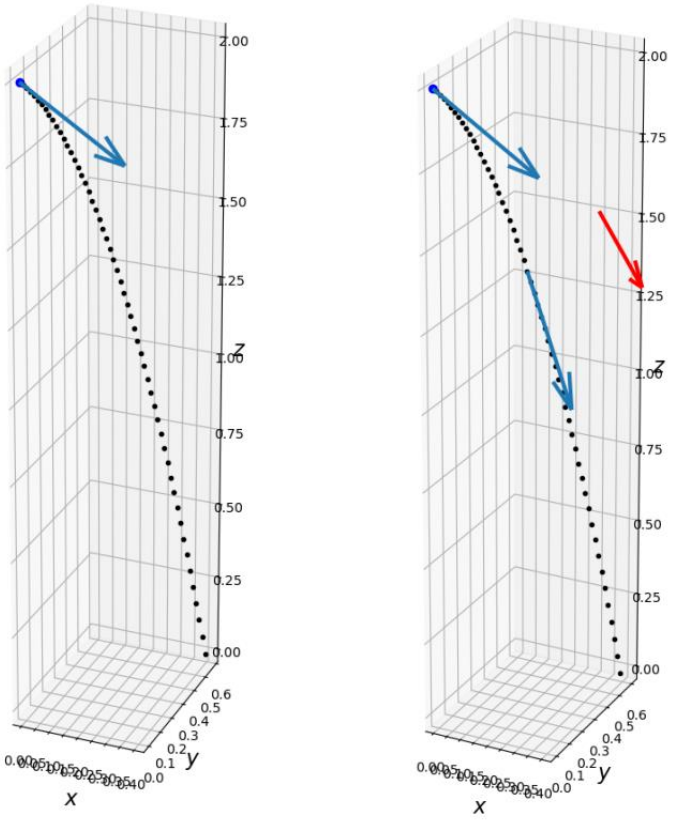

Figure 1 – Object trajectory with initial launching vector (Left), and hand vector and closest trajectory point with respective orientation (Right)

To be able to trace the trajectory path of the flying object, the simulation will send feedback through two error messages:

1) Distance: describing the distance, in meters, between the end effector and falling object trajectory. This error message will not specify the distance along x, y and z coordinates, but a generic distance measurement.

2) Angular Deviation: describing the rotational distance, in degrees, between the direction of the falling object and the end effector orientation. The error message will print a singular value, not specifying the individual angular disparity of Roll, Pitch and Yaw.

A third error message will display if the object has been caught or not.

The main challenge of the task is to develop an algorithm that, by adjusting the position and orientation of the manipulator based on the feedback received, will be able to catch every new object thrown, no matter the how diverse the new trajectories' parameters are.

Simulated Environment

This task will be carried out using tools and packages provided within a Virtual Machine (VM) running a distribution of Linux as operating system. The simulation and the Python scripts will be launched and operated using ROS (Robot Operating System).

The simulation environment will consist in an empty 3D space in which a UR10 robotic arm will be place and centred in a fixed point. The programmed movement of the manipulator will be visible through animations, although neither the flying object nor its trajectory will be shown in the simulation.

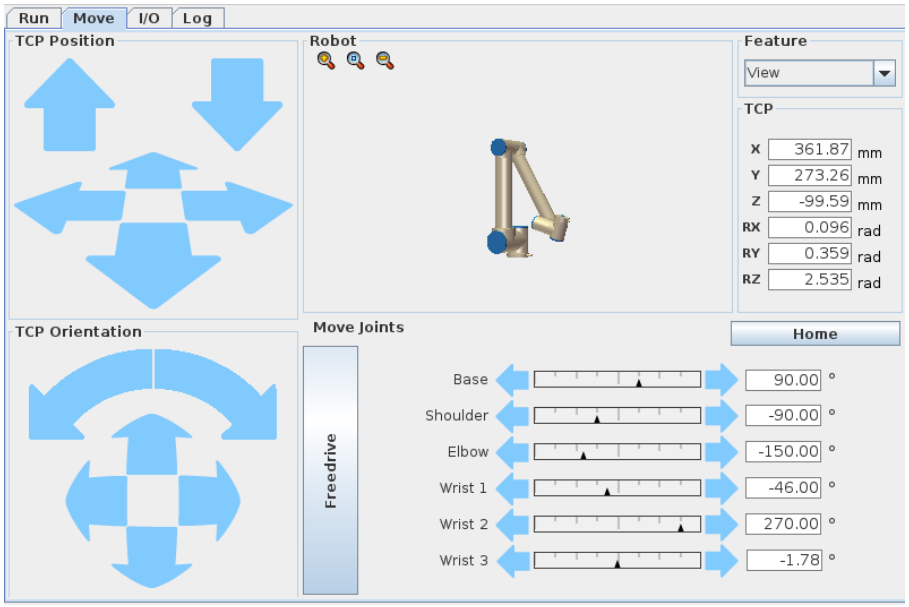

Figure 2 – UR10 Simulation window

The reference frame has its origin at the bottom of the UR10 arm, with the x axis positioned laterally to it, the y axis facing forward in relation to the robot, and the z axis placed vertically. The object's launching speed will vary between 0.1/1.5 m/s, its downward angle between 0°/45° and its lateral angle between -15°/15°. For the purpose of this task, any friction is ignored, and the acceleration is represented by 9.81 m/s^2

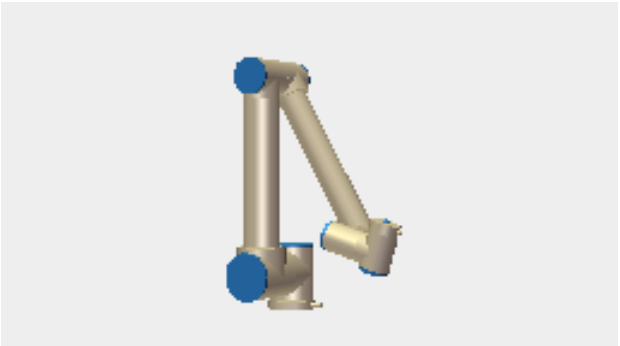

Figure 3 – UR10 virtual representation and coordinate frame

Algorithm

A thorough examination of the task's problem has been necessary, prior the draft of the algorithm. Due to the inability to visualize the object's trajectory within the simulation, the first issue was to identify from which direction the object was thrown. Although the UR10 must be able to find the trajectory regardless of its starting point and orientation, while initially troubleshooting the code and examining the efficiency of the used method, a smaller error was preferred.

Initially, with the default home waypoint given with the task, both distance and orientational error values were significantly large. By implementing Move-Linear (move_l) actions to decrease the distance error, the robot required an extended amount of movements, and in addition, before reaching the desire threshold value, the UR10's joints will often collide among themselves, breaking the simulation.

To reduce those accidents, I created 4 separate home waypoints where the robot's end effector faced in the positive X [Fig. 4], negative X [Fig. 5], positive Y [Fig. 6] and negative Y [Fig. 7] directions.

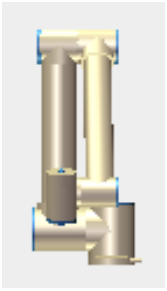

Figure 4 – UR10 Home Waypoint 1,
End Effector Facing Positive X Orientation

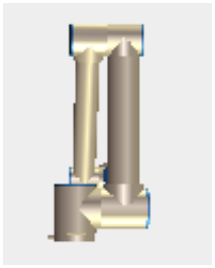

Figure 5 - UR10 Home Waypoint 2,
End Effector Facing Negative X Orientation

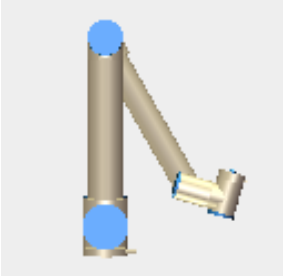

Figure 6 – UR10 Home Waypoint 3,
End Effector Facing Positive Y Orientation

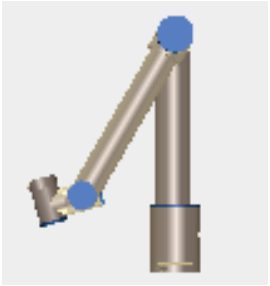

Figure 7 - UR10 Home Waypoint 4,
End Effector Facing Negative Y Orientation

Once the manipulator moved at each waypoint, the new error values were updated, and the home position which returned the smallest error, was then selected as starting location for the sequence. While performing those steps, it became clear that moving the UR10 with linear movements (move_l) was not optimal, in fact the robot's end effector often collided with its joints, once again terminating the simulation. By experimenting with the function 'move_j_pose()', which instead allows the manipulator to reach a specified position by non-linear movements, the collision issue was resolved.

I noticed that each time a new trajectory was created, the home position which returned the smallest error always resulted as Home Waypoint 3 [Fig. 6], consequently instead of using four different home positions, only that one has been use as starting position.

The next step was to create a solution to reduce both distance and rotational error. I decided to firstly reduce the distance error, and only once its value reached below the wanted threshold (0.05m), the rotational error of the end effector was rectified.

To start, a main while loop checking whether the object has been caught or not has been created. This will allow to stop the program execution once both distance and rotational errors have reached their threshold.

Originally, a single while loop has been implemented, running under the condition that the distance error value is greater than the desired threshold. If true, the manipulator will move sequentially along the X, Y and Z axis, slowly reducing the given error by executing the following sequence:

- Move the end effector of + 5 cm along the selected axis.
- If the new error given has a larger value than the previous, move 10cm the opposite direction (-0.10).
- If on the other hand, the resulting error has a smaller value than the previous one, proceed and perform the same sequence in the next axis.

Although this method resulted effective, for execution with initial error greater than 0.25m, the sequence took an extended amount of moves and time to reach the desired threshold value. To rectify this limitation, the movements values were increased, but as result the UR10 would continuously shift between two poses, unable to further reduce the error and interrupt the loop.

To optimize the solution, three while loops in total are used, all adopting the same methodology but with different conditions and movements values. Their specifications are displayed in Table 1 below.

While Loop Condition (distance error value)	Distance to move forward (+)	Distance to move backward (-)
> 0.25 m	+ 0.10m	-0.20m
> 0.10 m	+ 0.05m	-0.10m
> 0.05 m	+ 0.02m	-0.04m

Table 1 – While Loop Conditions and Distance Values

Once the distance error reaches a value smaller than 5cm, the program will print a message in the terminal, confirming that the desired threshold has been reached [Fig. 8], then exit from the last loop and proceed to the method used to decrease the angular deviation given by the rotational error.

Another method to reduce the errors received has been originally explored. Instead of moving the manipulator forward and backward depending on the new increased or decreased value obtained, the program would check, for each axis, in sequence, if the error was decreasing while moving either forward or backward, then proceed depending on the optimal movement direction. After an accurate analysis, this method has been discarded for two reasons:

1. To not repeat backward/forward movement each time the loop iterates, and by working on an axis at the time, reducing the error to the threshold resulted less accurate and more complex.
2. The code clarity was significantly decreased with this method, despite the use of functions.

```
MOVE:   Axis:  x        Distance:        -0.1
Distance Error:       0.13548170030117035
MOVE:   Axis:  y        Distance:        0.05
Distance Error:       0.08855076879262924
MOVE:   Axis:  z        Distance:        0.05
Distance Error:       0.07748255878686905
MOVE:   Axis:  x        Distance:        0.02
Distance Error:       0.07597452402114868
MOVE:   Axis:  y        Distance:        0.02
Distance Error:       0.059545595198869705
MOVE:   Axis:  z        Distance:        0.02
Distance Error:       0.04925656318664551
##### --- Distance Threshold Reached! --- #####
```

Figure 8 – ROS Terminal Displaying Printed Messages of Decreasing Distance Error Values, Until Threshold is Reached.

Unlike the previous approach, only one while loop is used to lower the rotational error value. While the error value is bigger than the 5°, the UR10's end effector will sequentially rotate along the X (Roll), Y (Pitch), and Z (Yaw) axes. The end effector will firstly rotate of + 0.1 radians; if the error given is larger than the previous, it will rotate in the opposite direction of -0.2 radians, otherwise the program will start the sequence in the next dimension of movement. Once the rotational error reaches a value below the 5° threshold, the program will exit the loop.

Afterwards, an if statement will double check if both error values have simultaneously met their threshold, and if true, the main loop checking for the object error status will break, and the algorithm will terminate. Additionally, before stopping the code execution, a string confirming that the ball has been caught will be displayed in the terminal [Fig. 9].

```
MOVE:   Dimension:  roll        Deviation:      -0.2
Rotational Error:       13.443238258361816
MOVE:   Dimension:  pitch       Deviation:      -0.2
Rotational Error:       14.329506874084473
MOVE:   Dimension:  yaw         Deviation:      0.1
Rotational Error:       13.652177810668945
MOVE:   Dimension:  roll        Deviation:      -0.2
Rotational Error:       8.569696426391602
MOVE:   Dimension:  pitch       Deviation:      0.1
Rotational Error:       7.134105205535889
MOVE:   Dimension:  yaw         Deviation:      -0.2
Rotational Error:       7.198061943054199
MOVE:   Dimension:  roll        Deviation:      -0.2
Rotational Error:       1.4711626768112183
THE BALL HAS BEEN CAUGHT
```

Figure 9 – ROS Terminal Displaying Printed Messages of Decreasing Rotational Error Until Threshold is Reached, and Ball Caught

Despite the high efficiency of this algorithm, it presents a limitation: if the forward movement/rotation return a higher error and the UR10 is then moved/rotated back of a negative value, the new error given might still be slightly higher than the previous one. Since the new error's values is only minimally larger, and by moving/rotating the UR10 forward again the results got will be very similar, the algorithm was considered efficient enough without further alterations.

For a better visualization of the algorithm sequence, a Flowchart [Fig. 10] has been added to this report.

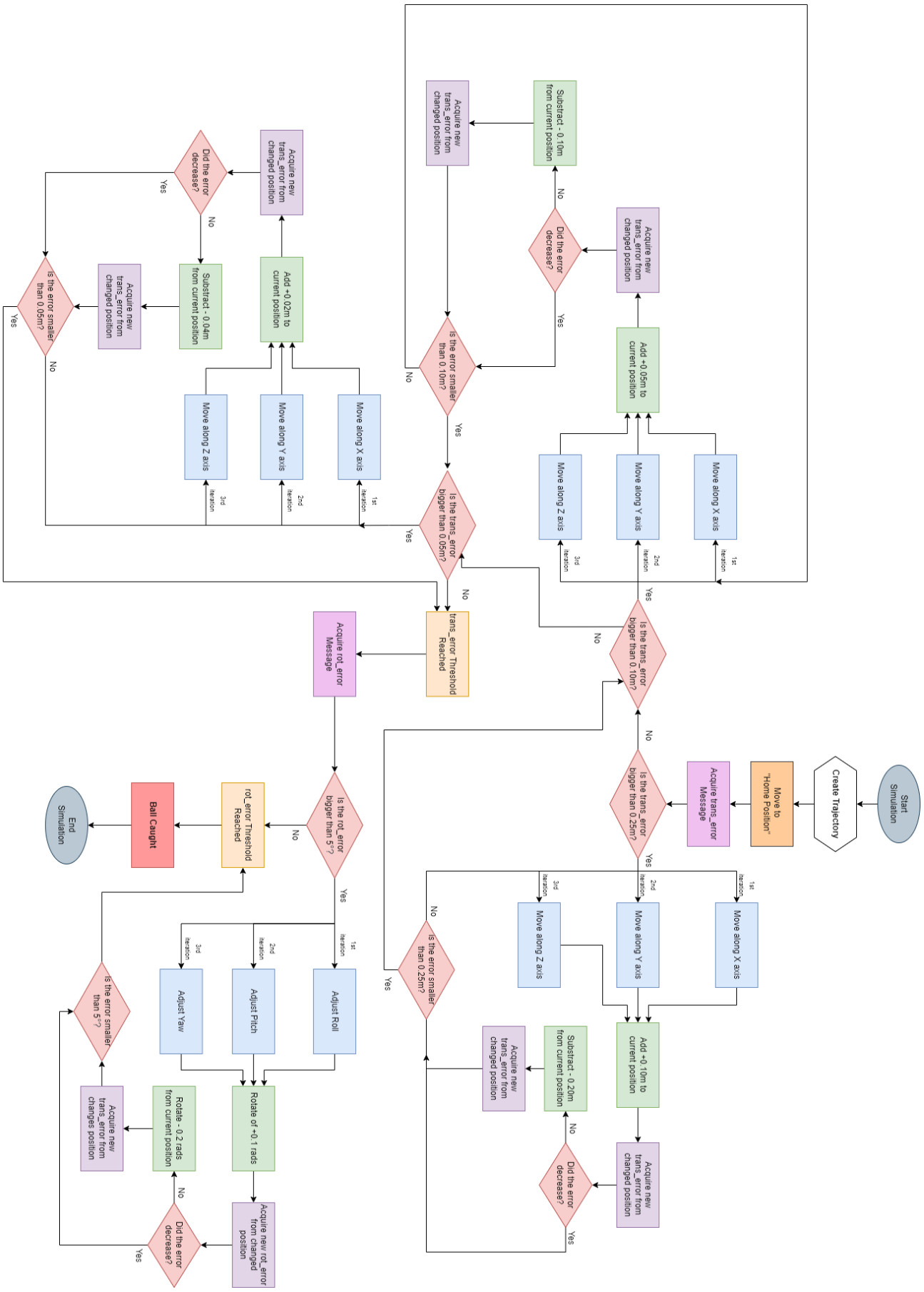

Figure 10 – Algorithm's Flowchart

Conclusion

The algorithm designed for this task, since fully developed, has been proven successful in 100% of the times in which it has been tested. The development of the method was backed by an extended analysis of the scripts initially provided, the application of heuristic method to familiarize with the program's syntax and logic flow, and a brief examination of approaches used in greedy algorithms [1], which inspired the logic composition of the code.

Reference

[1] S. A. Curtis, 'The classification of greedy algorithms', *Science of Computer Programming*, vol. 49, no. 1–3, pp. 125–157, 2003.