Technical Document

Simulation Pybullet

Index

Introduction to Problem statement

Dual-Arm Robot Coordination

- **Robot A (UR5)** and **Robot B (Franka Emika Panda)** are mounted on separate tables facing each other.
- Robot A picks up an object in its workspace and places it at a target location.
- Then, Robot A moves the object along **circular** and **Lissajous** trajectories, switching between them periodically.
- Robot B, with a **camera on its end-effector**, tracks the object visually and follows its motion using image-based visual servoing.

Process of development and Flow Chart

Initially after understanding the problem statement, following needs to be sorted :

1) Simulation environment
2) URDF files for robot and environment (Tables and ball)

3) Inverse Kinematics and Joint Control
4) Tracking Algorithm
5) Programming Language

Simulation Environment :

Gazebo and PyBullet were initially considered based on prior experience. After evaluating the system requirements, **PyBullet was selected** for the following reasons:

- **Lightweight** compared to Gazebo, making it more suitable for quick testing and development.
- **ROS 2 integration is not essential** for this simplified setup.
- **Custom joint control** is a key goal, and relying on prebuilt controllers (as in Gazebo/ROS) would not align with the objective of developing and testing self-written trajectory and control logic.

URDF files for robot and environment

URDF files were available in Git Repo in open source and thus were used but they were for **ROS format and had to be converted to be usable for Pybullet.**

Inverse Kinematics and Joint Control

A simulation scene was created in PyBullet to explore inverse kinematics (IK) and joint-level control of a robot. The concepts were understood through official documentation, reference code, and practical experimentation. Simple scripts were developed to control the robot using both inverse kinematics and direct joint manipulation. After establishing basic motion control, additional tasks such as applying constraints and linking external objects to the robot were successfully implemented.

```
pybullet.calculateInverseKinematics(robot,
self.end_effector_index_ur5, position, quaternion,
     jointDamping=[0.01]*6, upperLimits=upper_limits,
     lowerLimits=lower_limits, jointRanges=joint_ranges,
     restPoses=current_joints
 )
```

All relevant joint parameters had to be understood, applied, and tested. This included setting **upper and lower joint limits**, defining valid joint ranges, and specifying rest poses to guide the inverse kinematics (IK) solver. **The use of rest poses helped reduce redundancy in solutions and ensured more stable and predictable robot motions within the defined constraints.**

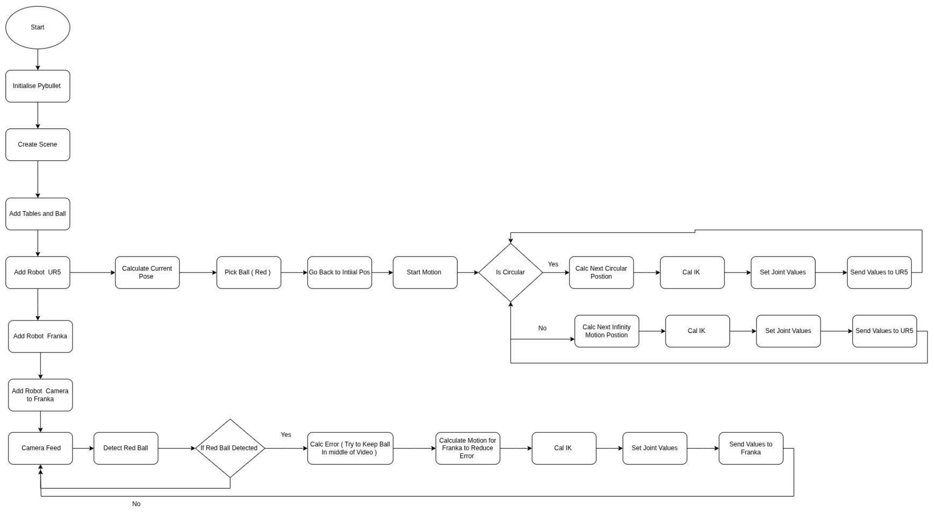
Tracking Algorithm

To keep object tracking simple and effective, a red ball was chosen as the target due to its distinct color, which made detection both straightforward and reliable. The ball's position is detected in each video frame, and the error is computed as the distance (in pixels) between the ball's center and the center of the image frame. This pixel-based error is then mapped to robot motion commands. After experimenting with multiple approaches, a proportional scaling method—multiplying the **pixel error by a constant**—was found to be the simplest and most effective way to convert image-space error into motion in the robot's X, Y, and Z axes.

Programming Language

Python language was selected for quick development.

Flow Chart of the code



Major Achievements

1) Understanding the API for Inverse Kinematics and Joint space control of robots in Pybullet.
2) Getting Pybullet with scene and robots (due to issues with URDF being only ROS compliant)
3) Understanding constraints on objects and linking them (why ball might roll over when placed on table !!!)
4) Multi threaded architecture, making it modular and scalable.

Tracking Performance

The tracking seems to be stable with robot continuously monitoring position at 30fps.

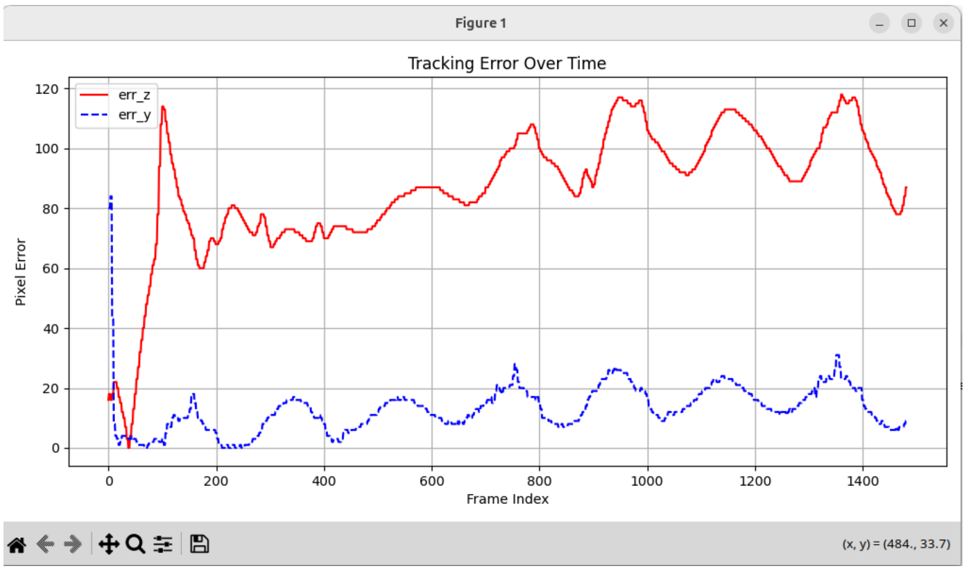


Figure 1
Tracking Error Over Time
err_z
err_y
Pixel Error
Frame Index
0
20
40
60
80
100
120
0
200
400
600
800
1000
1200
1400
(x, y) = (484., 33.7)