Final Report: Introduction to Robotics Lab

S M. Ali Naqvi*, and Zaid Bin Khalid †
Habib University, Pakistan
Email: *sn07590@st.habib.edu.pk, †zk08128@st.habib.edu.pk

***Abstract*—This project presents the design and development of an autonomous robotic system capable of executing pick-and-place tasks using a 4-DOF PhantomX Pincher manipulator. The system is engineered without reliance on high-level frameworks such as ROS, focusing instead on modular implementation of perception, planning, and control in MATLAB. A custom RGB-D perception pipeline performs color-based segmentation and 3D pose estimation using point cloud data and Iterative Closest Point alignment. Symbolic forward kinematics are derived using screw theory, and an analytical inverse kinematics solver computes feasible joint configurations. Motion is governed by a Finite State Machine (FSM), while trajectory execution is optimized using Jacobian-based rate control. Experimental results demonstrate reliable object detection, accurate pose estimation with a mean absolute error of 2.13 cm, and robust performance across multiple object placements. This work presents a classical robotics design optimized for constrained hardware, and establishes a foundation for future enhancements such as real-time feedback control and adaptive grasp planning.**

***Index Terms*—Phantom X Pincher, Robot Manipulator, Pick and Place**

I. Introduction

Robotic manipulation plays a critical role in automation across manufacturing, logistics, and service industries. Robotic arm manipulators have become the cornerstone of modern automation, enabling machines to interact with their environment in structured and intelligent ways. These mechanical systems, often inspired by the kinematics of the human arm, are capable of executing a wide range of tasks such as assembly, welding, painting, packaging, and more recently, complex interactions in unstructured environments like warehouses, homes, and healthcare facilities. Among the most fundamental and widely studied applications of robotic manipulators is the pick-and-place task , the process of identifying, grasping, transporting, and placing objects from one location to another with precision and reliability.

The pick-and-place operation, while seemingly simple, encapsulates several challenging problems in robotics, including perception, motion planning, control, and manipulation. Successfully performing this task requires a tightly integrated pipeline that connects a robot's sensory inputs to its actuation commands through robust algorithms and control strategies. This project aims to develop such a pipeline for an arm manipulator, with a focus on enabling it to autonomously perform pick-and-place operations in a controlled environment.

The aim of this project is to build a pipeline for robotic pick-and-place task using the PhantomX Pincher, a 4-DOF articulated arm, integrating object detection and localization, trajectory planning, inverse kinematics, and motion execution , all tailored to the physical constraints and degrees of freedom of the specific robotic arm in use without relying on any high level libraries or frameworks like ROS2 moveit etc. The core objective was to enable the robot to detect, localize, and move colored objects within its workspace using RGB-D perception and perform accurate pick and placement tasks.

The significance of this work lies in its application of core robotics principles, color-based segmentation, 3D pose estimation, and inverse kinematics, to create an autonomous system capable of performing complex manipulation tasks. Through a semester-long exploration of theoretical concepts and hands-on implementation, the project simulated real-world robotic workflows involving sensor integration, feedback loops, and control architectures.

The project relies on the Intel RealSense depth camera to provide RGB-D input and Phantom X Pincher 4-DOF Arm for task execution. The project is built using MATLAB as the programming framework. For interfacing with the robot, Peter Cork's robotics toolbox is used. For interfacing with the Intel real sense camera, intel real sense MATLAB pipeline is used.

II. Literature Review

Robotic pick-and-place tasks are a fundamental capability in automation and manufacturing, and a broad range of methodologies have been explored to improve accuracy, adaptability, and ease of deployment. These approaches range from classical control and kinematics-based methods to more recent strategies leveraging learning and human demonstration.

Lobbezoo *et al.* provide a comprehensive survey of reinforcement learning (RL) applications in robotic pick-and-place tasks, emphasizing its potential to eliminate the need for manual programming by enabling robots to autonomously learn task-relevant behaviors through trial and error [1]. This approach allows for dynamic adaptation in environments that are difficult to fully model. However, RL methods often demand significant data and tuning to be effective, and real-world generalization remains a challenge.

Complementing the learning-based perspective, Skoglund *et al.* discuss programming by demonstration (PbD), where a robot learns pick-and-place sequences from human-guided examples using motion primitives [2]. This strategy reduces the need for domain-specific programming expertise and supports intuitive training, but may struggle with robustness and task generalization without additional optimization layers.

In a related effort, Lin and Chiang propose a gesture-based interface that enables robots to learn pick-and-place actions through observed human hand gestures using convolutional neural networks (CNNs) for gesture recognition and a behavior-based programming platform (XABSL) for task

execution [3]. Their system achieves high recognition accuracy and demonstrates how gesture-driven interaction can enhance natural human-robot collaboration in task training.

Energy-efficient task planning is another critical axis explored in the literature. Pellicciari *et al.* propose optimizing task execution time (TET) for existing industrial manipulators by modeling electromechanical properties and applying time-scaling to reduce energy consumption during pick-and-place cycles [4]. This work emphasizes sustainability and operational cost reduction, which are particularly relevant for large-scale deployments.

On the perception side, Kumar *et al.* present a vision-based object detection and recognition system designed to identify, classify, and localize objects in the workspace using feature extraction and artificial neural networks (ANNs) [5]. Their system achieves high recognition accuracy but requires careful tuning of classifiers and pre-processing algorithms, particularly in cluttered environments.

Lastly, Harada *et al.* introduce an object placement planner that selects stable placements by analyzing contact surfaces and ensuring static equilibrium using geometric models derived from point clouds [6]. This approach is useful in unstructured environments where safe and stable object positioning is a priority.

Compared to these methods, our approach employs classical kinematics (forward and inverse) with RGB-D-based segmentation and pose estimation, combining reliability with ease of implementation. While it does not leverage machine learning or imitation-based training, it offers transparency, control, and sufficient accuracy for structured lab environments. Our system prioritizes robustness and interpretability over adaptiveness, making it suitable for well-defined industrial or academic settings where repeatability is key.

III. Design Details

This section outlines the architecture and key components of the robotic pick-and-place system, including perception, kinematics, motion planning, and control. The system was built incrementally, with each module tested in simulation and on hardware before integration.

A. System Architecture

The robotic system comprises the following subsystems:

- **Perception Module**: RGB-D sensing and color-based object segmentation
- **Pose Estimation**: 3D localization of colored cubes using point clouds
- **Forward Kinematics**: Computation of end-effector pose from joint angles
- **Inverse Kinematics**: Analytical solver to find joint angles from target poses
- **FSM Controller**: Finite state machine for managing robot behavior

In the subsequent section, we describe our system components in details

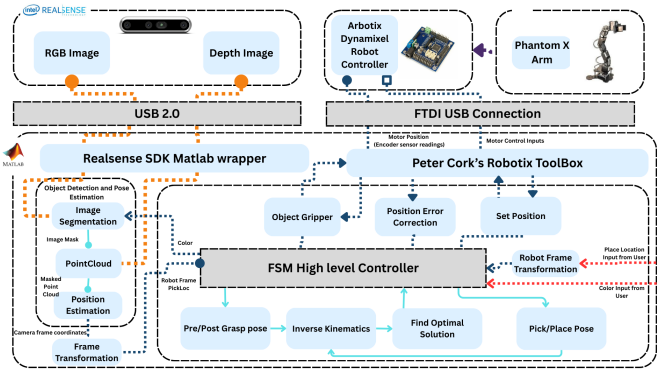

Fig. 1. The figure shows an overview of the system for the Pick Place Pipeline

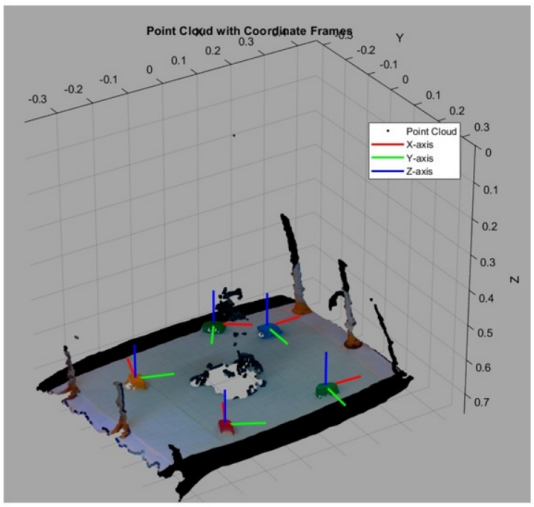

Fig. 2. Frame assingment for all detected objects.

B. Object Detection and Pose Estimation

To segment colored cubes, we employed a two step procedure comprising of a color based mask follwoed by a depth mask. The color mask is first used to approximately identify the position of the objects in the pixel (image) space. We used a combination of HSV and LAB color spaces for this step. Masks were generated using thresholding and refined using morphological operations (`imfill`, `bwareaopen`, `imclose`). Then, this RGB mask is converted to real-world coordinates, overlayed on the depth point cloud extracted from the Depth Data to further refine the identification of only the top surface of the cube for enhanced pose estimation. Once the masked point cloud is refined using depth data, the rest of the region is masked out. The top surface is then extracted using MATLAB's in built plane fitting algorithm `pcfitplane`. Using a prebuilt geometric model of the cube, we estimate the cube's position and orientation using ICP. High level algorithm is given below. Our results show centimeter level accurate position estimation as shown in figure 1. A detailed error analysis is given in Position Estimation Lab Report. [].

Algorithm 1 High-Level Pipeline for Cube Detection and Pose Estimation

function DETECTANDESTIMATEPOSE(color)
  Acquire RGB and Depth Frames from Camera
  Load Cube Face Model Point Cloud
  Apply Color-Based Segmentation using HSV and LAB spaces
  Refine Masks with Morphological Operations (e.g., `imfill`, `imclose`)
  Convert RGB Masks to 3D Point Clouds using Depth Data
  Initialize empty list of transformations
  for all Detected Objects **do**
    Generate Point Cloud using Color Mask
    Fit and Remove Dominant Plane from Scene (`pcfitplane`)
    Remove Black/Empty Background Points
    Extract and Flatten Top Surface Plane of the Cube
    Downsample Model and Scene Point Clouds
    Estimate Pose using ICP between Model and Extracted Top Surface
    Store Transformation Matrix
  end for
  return RGB Segmentation Overlay and List of Transformations
end function=0

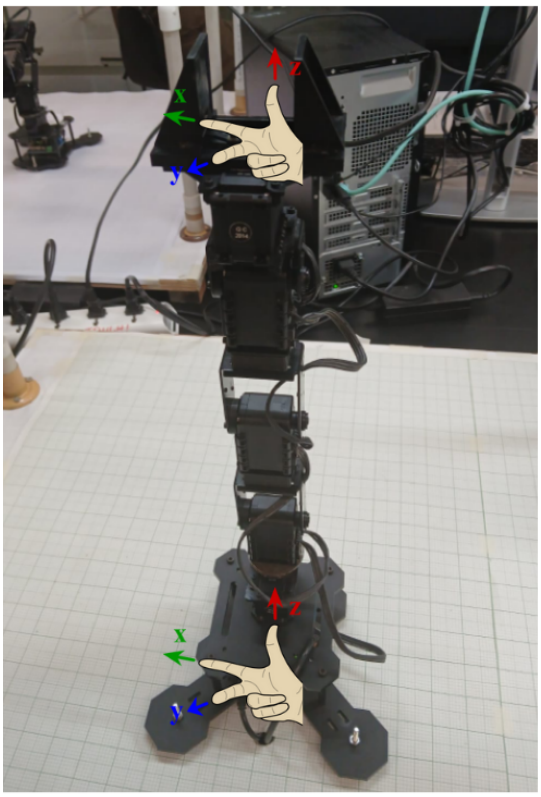

Fig. 3. Frame assingment for Robotic Arm.

C. Frame Assignments and Forward Kinematics

We first assign the camera frame, world frame and the robot frame. All three frames have positive x and y in the same direction which means there is no relative rotation between them. The camera frame has an offset on the z-axis while the robot frame has an offset in x-y plane from the world frame. The world frame is taken to be at the center of the workspace. These frames are shown in figure.

For the end effector frame, we use derive its forward kinematics equation. Home pose was choose to be the same when servo angles are set to zero which is the upright position of the robot and using screw theory, the forward kinematics were derived symbolically using the product of exponentials formula. Screw axes were defined based on the joint axes and link configurations. The symbolic transformation matrix T_0^4 was computed and verified using both simulation and real arm measurements. Verification results can be found in Forward Kinematics lab [].

D. Inverse Kinematics and Optimal Solution

1) Inverse Kinematics Formulation: The inverse kinematics (IK) problem for the 4-DOF Pincher robot is to determine the joint angles $\theta_1, \theta_2, \theta_3, \theta_4$ that place the end effector at a desired position (x, y, z) with orientation ϕ (in the xz-plane).

The arm link lengths are known as:

$$\begin{aligned} L_{12} &= 14.1 \text{ cm} \\ L_3 &= 10.6 \text{ cm} \\ L_4 &= 10.6 \text{ cm} \\ L_5 &= 7.2 \text{ cm} \end{aligned}$$

Let:

$$\begin{aligned} r &= \sqrt{x^2 + y^2}, \\ s &= z - L_{12}, \\ u &= r - L_5 \cos(\phi), \\ v &= s - L_5 \sin(\phi) \end{aligned}$$

These transformations account for the projection of the desired position onto the arm's reachable workspace, compensating for the offset caused by the final link L_5 and orientation ϕ.

a) Joint Angle Calculations::

1) **Base rotation θ_1:**

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) \quad \text{and} \quad \theta_1 = \pi + \tan^{-1}\left(\frac{y}{x}\right)$$

These represent two symmetric configurations around the base axis.

2) **Elbow joint θ_3:** Using the law of cosines:

$$D = \frac{u^2 + v^2 - L_3^2 - L_4^2}{2L_3L_4}, \quad \theta_3 = \tan^{-1}\left(\frac{\sqrt{1 - D^2}}{D}\right)$$

Both elbow-up and elbow-down configurations are considered.

3) **Shoulder joint θ_2:**

$$\theta_2 = \tan^{-1}\left(\frac{v}{u}\right) - \tan^{-1}\left(\frac{L_3 \sin(\theta_3)}{L_3 + L_4 \cos(\theta_3)}\right)$$

Algorithm 2 Forward Kinematics for 4-DOF Pincher Robot

Require: Joint angles $\theta_1, \theta_2, \theta_3, \theta_4$

Ensure: End-effector position (x, y, z) and pose matrix R

1: Define link lengths: $L_{12} = 14$, $L_3 = 10.6$, $L_4 = 10.6$, $L_5 = 7.4$

2: Define rotation axes: $w_1 = [0, 0, 1]^T$, $w_{4\_2} = [1, 0, 0]^T$

3: Define joint positions:

- $q_1 = [0, 0, 0]^T$
- $q_2 = [0, 0, L_{12}]^T$
- $q_3 = [0, 0, L_{12} + L_3]^T$
- $q_4 = [0, 0, L_{12} + L_3 + L_4]^T$

4: Compute linear velocity vectors:

- $v_1 = -w_1 \times q_1$
- $v_2 = -w_{4\_2} \times q_2$
- $v_3 = -w_{4\_2} \times q_3$
- $v_4 = -w_{4\_2} \times q_4$

5: Construct screw axes:

- $S_1 = [w_1; v_1]$
- $S_2 = [w_{4\_2}; v_2]$
- $S_3 = [w_{4\_2}; v_3]$
- $S_4 = [w_{4\_2}; v_4]$

6: Define home configuration matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_{12} + L_3 + L_4 + L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

7: Compute transformation matrices using matrix exponential:

$$T_{01} = \exp(S_1 \cdot \theta_1)$$
$$T_{12} = \exp(S_2 \cdot \theta_2)$$
$$T_{23} = \exp(S_3 \cdot \theta_3)$$
$$T_{34} = \exp(S_4 \cdot \theta_4)$$

8: Compute full transformation:

$$T_{04} = T_{01} \cdot T_{12} \cdot T_{23} \cdot T_{34}$$

9: Compute end-effector pose: $R = T_{04} \cdot M$

10: Extract position: $x = R_{14},\ y = R_{24},\ z = R_{34}$

11: **return** (x, y, z, R)

=0

4) **Wrist orientation θ_4:**

$$\theta_4 = \phi - \theta_2 - \theta_3$$

This results in two candidate sets of joint solutions $Q = \{\mathbf{q}_1, \mathbf{q}_2\}$.

2) Joint Angle Normalization and Filtering: Each angle is normalized to the range $[-\pi, \pi]$ using:

$$\text{normalize}(\theta) = \mod(\theta + \pi, 2\pi) - \pi$$

Only solutions within the joint limits $[-150^\circ, +150^\circ]$ are retained.

3) Optimal Solution Selection: The optimal solution is chosen by minimizing the weighted Euclidean distance from the current robot configuration $\mathbf{q}_0$ to each valid candidate solution $\mathbf{q}_i$:

$$\mathbf{q}_{\text{opt}} = \arg \min_{\mathbf{q}_i \in Q_{\text{valid}}} \sum_{j=1}^{4} w_j (\theta_{ij} - \theta_{0j})^2$$

where:

- θ_{ij} is the j^{th} joint angle in candidate solution i
- θ_{0j} is the j^{th} joint in the current configuration
- w_j are weights for each joint, e.g., $[1, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]^T$ to prioritize base joint movement

The solution with the minimum cost is selected as the final output:

$$\mathbf{q} = \mathbf{q}_{\text{opt}}$$

E. PreGraps Pose Selection

We divide our circular workspace in two regions. The lower and higher region. The lower region is where the robot can perform pick place task with its end effector facing straight down, and the higher one was where the robot could reach only reach if its end effector is titled outwards from itself. Subsequently, we define two pregrasp and grasp poses based on the part of workspace the objects and desired place points are at. The algorithm is given below.

Algorithm 3 `preGraspPose` – Compute Pre-Grasp Pose

0: **function** PREGRASPPOSE(pose_obj)
0: $x \leftarrow$ pose_obj[1]
0: $y \leftarrow$ pose_obj[2]
0: $z \leftarrow$ pose_obj[3]
0: **if** $\sqrt{x^2 + y^2} < 18.32$ **then**
0: pose $\leftarrow [x;\ y;\ z + 5;\ -\pi/2]$
0: **else**
0: pose $\leftarrow [x;\ y;\ z + 5;\ -45]$
0: **end if**
0: **return** pose
0: **end function**=0

F. Position Controller

Once Optimal Solution for a specific position is determined, we designed a position controller to move to the desired cartesian position. The position controller is designed to ensure the manipulator has actually reached the desired angles command the robot was fed by checking the current position at a fixed frequency rate. Moreover, it also employs several checks to ensure that the velocity is within safe limits and the desired angles is within the limitsof the servo motors.

1) Error Correction for Encoder: One problem we identified with our Phantom X arm was that there was an error in the first servo form the base. At $\theta = -\pi$, it overshot, while the encoders still reported the angle as $-\pi$. We saw that this error increased linearly when θ went from π to $-\pi$. therefore, we designed a linear error correction module for encoder position correction to ensure correct position tracking in trajectory.

The corrected joint angle $\theta_{\text{corrected}}$ is computed as:

$$\theta_{\text{corrected}} = \theta + \left(\frac{0.0524}{\pi}\theta - \frac{0.0524}{2}\right) \quad (1)$$

Simplifying the expression:

$$\theta_{\text{corrected}} = \theta\left(1 + \frac{0.0524}{\pi}\right) - \frac{0.0524}{2} \quad (2)$$

where $\theta = \texttt{jointAngles}(1)$ is the original joint angle.

G. Iterative Position Error Correction

With prolonged use, the robot sometimes fails to reach the correct pick/drop location. Other times, the perception pipeline falls in some edgecases producing small devations in the perceived center points of the cube vs the actual center point. To overcome these issues, we designed an iterative position error correction module which rectifies the deviation in case the robot fails to go to the exact grip location, failing to grip the object. When the endeffectos fails to reach the center point of the object, one of the endeffector jaws end up on top of the cube. This module sense when the robot jaw gets stuck on top of the cube. When this happens, the end effectors jaws are moved which produce a movement in the base servo joint. The motor encoders are then used ot read this movement to determine the direction the arm needs to move to correct its location. The robot iteratively corrects its location until it ensures successful picking up of the object.

State Update: The updated base joint configuration q_1 is computed iteratively as:

$$q_1 \leftarrow q_1 + (\theta_{\text{corrected}} - q_1)$$

Iterative Correction Loop: The correction continues until the object is successfully picked:

$$\texttt{while success} = -1 \texttt{ do}$$

Read current joint position: $\theta_{\text{raw}} = \texttt{getEncoderReading}()$

Apply correction: $\theta_{\text{corrected}} = \theta_{\text{raw}} - \left(\frac{0.0524}{\pi} \cdot \theta_{\text{raw}} - \frac{0.0524}{2}\right)$

$$q_1 \leftarrow q_1 + (\theta_{\text{corrected}} - q_1)$$

$$\texttt{success} \leftarrow \texttt{attemptPick}(q_1)$$

$$\texttt{end while}$$

H. Object Gripper

Once the object is at the pick/drop location, the object gripper module picks/drops the object using feedback from the encoders to ensure the object has been picked/dropped. This module translates the angle of the servo to the linear distance of the jaws and uses a fixed distance (length of the cube) to check if the object is picked/dropped. This information is essential for successful execution of the task and is used in the feedback mechanism for position correction (in case of incorrect position).

Algorithm 4 Grip Object Based on Servo Feedback

```
function GRIPOBJECT(position)
    if position is outside valid range then
        return FAILURE {Invalid object location}
    end if
    Compute grip offset angle based on object dimensions
    Calculate gripper jaw angle using:
      θ = (max_position − position) / Δ
    Add grip offset to calculated angle
    Command gripper to move to resulting angle
    Wait briefly for movement to complete
    Read actual joint angle from sensor/encoder
    if actual angle deviates from expected range then
        return FAILURE {Grasp likely unsuccessful}
    end if
    return SUCCESS
end function=0
```

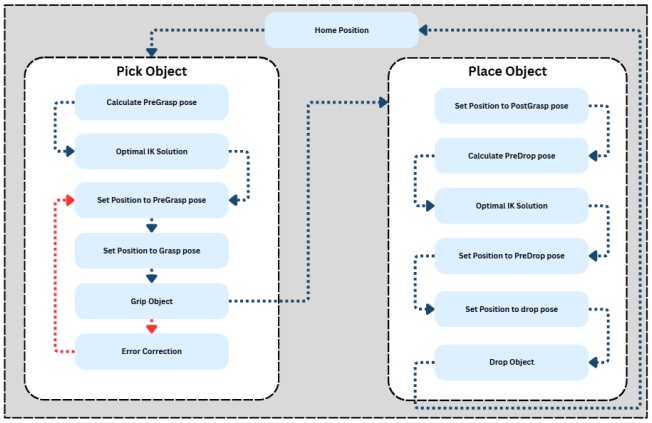

Fig. 4. FSM for task execution. The red arrows indicate flow in the case of failure.

I. Motion Planning and FSM

With the modules in place, we design a high-level controller that utilizes the modules for handling the input and controlling the robot for successful execution of the task. When an input is given, it sets the robot to its home position first, then it takes the object location from the object detection module and computes the pregrasp pose. With the pregrasp pose, it finds the optimal set of servo angles for the shortest and safest path. After ensuring successful execution to that point, it transitions to the grasp pose. In most of the cases It grasps the object in first attempt. HOwever in some cases one of the jaws coincides with the face of the cube. The robot uses the position correction module to iteratively correct its position based on the direction the base servo has moved when it tries to close its gripper while in contact with the face of the cube. Once it picks up the cube, it goes to its pre drop location, avoiding any other cubes, and finally places/stacks the cube in the desired location/bin. The following figure shows the High Level FSM for the motion control.

IV. Results

In the results section we present an evaluation of the robot's performance across perception, kinematics, and motion control. Both quantitative metrics and qualitative observations are discussed to assess the system's accuracy and robustness.

A. Success Metrics

System performance was evaluated using the following criteria:

- **Pose Estimation Accuracy**: Euclidean distance between computed and measured end-effector positions.
- **Manipulation Success**: Ability to complete pick-and-place cycles without failure or collision.
- **Trajectory Fidelity**: Comparison of planned vs. actual joint trajectories and end-effector paths.
- **Robustness**: Tolerance to varying object positions and grasp orientations.

B. Pose Estimation Accuracy

Validation against physical measurements showed. Note that these tests were carried out BEFORE implementation of the Encoder Error Correction module:

TABLE I
Pose Estimation Accuracy Results

Test Point	Measured Position (cm)	Computed Position (cm)	Error (cm)
p1	(13.84, 13.08, 17.4)	(15.00, 15.00, 20.00)	3.03
p2	(-15.24, -11.18, 19.0)	(-15.00, -10.00, 20.00)	1.11
p3	(0.00, 12.45, 32.60)	(0.00, 12.00, 34.00)	1.82
p4	(-20.83, 11.18, 16.5)	(-20.00, 12.00, 20.00)	3.56
p5	(10.29, -10.67, 13.00)	(10.00, -10.00, 14.00)	1.11

Mean Absolute Euclidean Error (MAE): **2.13 cm**
Observation: Highest errors were in the Z-axis, likely due to limited servo resolution and measurement variability.

C. Motion Control Performance

- **FSM Execution**: The pipeline was tested several times for more than 10 boxes of different color. Tasks like bin picking as well as cube stacking were executed with different color and position inputs. The pipeline demonstrated successful transitions, executions as well as state recovery upon failure.
- **Pick-and-Place Cycles**: Multiple runs were completed with varying pick/place positions; the robot consistently grasped and released objects within ± 0.5 cm of the intended location.

D. Trajectory Planning & Execution

- Velocity control using Jacobians and symbolic expressions produced smooth joint motion.
- Simulated vs. actual end-effector trajectories were plotted for validation:
  - **Simulated path**: Direct, smooth arc
  - **Actual path**: Slight deviation but convergent, especially in Z-axis
- Errors and delays primarily occurred due to:
  - Joint angle quantization (servo limits)
  - Singular configurations near arm's stretched state

E. System Robustness and Edge Cases

- **Singularity Detection**: Rank loss was observed when $\theta_3 \approx 0$; velocity-based controllers returned large values.
- **Error Handling**: Limits enforced via `setPosition` avoided motor overrun and out-of-bound configurations.
- **Physical Constraints**: Object location too close to the arm base caused occasional grip misalignment.
- Long-time-use errors: When the robot ran for longer periods of time, the position error kept increasing. However, the position correction module corrected the error itertively to ensure successful task execution.

V. Conclusion and Future Work

This project successfully integrated perception, kinematics, and control to build a complete robotic pick-and-place system using the PhantomX Pincher arm. Over the course of the semester, a robust pipeline was developed to detect colored cubes using RGB-D data, compute their 3D poses, and manipulate them using forward and inverse kinematics-based control strategies.

The final system demonstrated a high level of accuracy and reliability, with a mean absolute Euclidean error of 2.13 cm in end-effector positioning. The Finite State Machine (FSM) effectively managed the execution of complex motion sequences, including object pickup, transfer, and placement. Rate control and velocity-based planning provided smooth trajectory generation, while modular error handling ensured safe and bounded execution even under edge-case scenarios.

From this project, several important lessons were learned. Integrating perception with motion control in real hardware environments involves inaccuracies such as sensor noise, servo resolution limits, and physical misalignments. Robust software design, modular control strategies, and careful calibration were key to managing these challenges.

In terms of the future improvements, below are the proposed ideas:

- **Closed-Loop Control**: Implementing feedback from vision or force sensors could improve precision and adaptability during manipulation.
- **Dynamic Obstacle Avoidance**: Integrating environment mapping and path planning would allow more dynamic and autonomous behaviors.
- **Real-Time Execution**: Transitioning from offline computation to real-time trajectory planning could enhance the system's responsiveness.

This project laid the foundation for advanced manipulation tasks and provided hands-on experience with real-world robotic system development.

Acknowledgments

The authors would like to thank the lab Research Assistant Sir Khuzaima Ali Khan for his valuable guidance throughout the project and Dr. Basit for their insightful feedback and continuous support. Their mentorship greatly contributed to the successful completion of this work.

PROJECT REPOSITORY

This work can be found at https://github.com/muhammadali74/PincherX_PickPlace_Package

REFERENCES

[1] A. Lobbezoo, Y. Qian, and H.-J. Kwon, "Reinforcement Learning for Pick and Place Operations in Robotics: A Survey," *Robotics*, vol. 10, no. 3, p. 105, 2021.

[2] A. Skoglund, B. Iliev, B. Kadmiry, and R. H. Palm, "Programming by Demonstration of Pick-and-Place Tasks for Industrial Manipulators Using Task Primitives," in *Proc. IEEE Int. Conf. on Robotics and Automation (CIRA)*, 2007.

[3] H.-I. Lin and Y. P. Chiang, "Understanding Human Hand Gestures for Learning Robot Pick-and-Place Tasks," *Int. J. Adv. Robot. Syst.*, vol. 12, 2015.

[4] M. Pellicciari, G. Berselli, F. Leali, and A. Vergnano, "A Method for Reducing the Energy Consumption of Pick-and-Place Industrial Robots," *Mechatronics*, vol. 23, no. 3, pp. 326–334, 2013.

[5] R. Kumar, S. Kumar, S. Lal, and P. Chand, "Object Detection and Recognition for a Pick and Place Robot," *The University of the South Pacific*, 2019.

[6] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, and H. Onda, "Validating an Object Placement Planner for Robotic Pick-and-Place Tasks," *Robotics and Autonomous Systems*, vol. 62, no. 10, pp. 1463–1477, 2014.