King's College London

Robotic Medical Assistant

Group Members: Zou Han | Shichao Zhang | Donghyun Ko

Student Name: Zou Han

Student ID: 24012533

Email: K24012533@kcl.ac.uk

2025-4

1. INTRODUCTION

With the rapid development of medical robotics, these technologies have shown great potential in improving surgical accuracy, reducing the workload of medical staff, and enhancing patient care[1]. Especially in the context of infectious disease prevention, reducing human contact, minimizing cross-contamination, and improving automation have become key goals[2]. Tasks such as assisting mobility-impaired patients, handling biological samples in labs, and delivering sterile instruments during surgery all require safe, precise, and collaborative robotic systems[3].

This project aims to develop an intelligent pick-and-place system based on vision recognition and robotic arm control, focusing on three typical medical scenarios:

1) object delivery for mobility-impaired patients.
2) sample transportation in labs.
3) handing over sterile tools in surgeries.

The system integrates ArUco marker detection[4], pixel-to-robot coordinate conversion[5], inverse kinematics, and real-time gripper feedback. To ensure safe operation, obstacle avoidance is also implemented.

The system uses OpenCV for camera calibration and marker detection, then maps pixel data to robot coordinates for path planning. A 3D-printed gripper with current-based feedback enables secure grasping. A GUI displays live camera feed and robot joint states, improving user interaction. Overall, this project demonstrates the integration of multiple key technologies in medical automation and provides a foundation for future applications.

2. METHODS

2.1 Experimental Setup

In this project, a vision-guided robotic system was built to perform automated pick-and-place tasks. The system consists of two Dynamixel motors to control the robotic arm joints and one motor for the gripper. A USB camera is fixed above the workspace to detect ArUco markers and locate target objects. The system is controlled by a Raspberry Pi 4 running ROS2, which manages communication between the vision module, control algorithms, and hardware interface. All components are mounted securely on a flat table, and the environment is kept well-lit to ensure stable marker detection and robot performance. The full hardware setup is shown in Appendix Figure A.

Component	Description
Robotic Arm	2× Dynamixel XC330-M288-T motors
Gripper	1× Dynamixel motor, controlled by current feedback
Camera	USB camera, top-mounted for ArUco detection
Controller	Raspberry Pi 4 running ROS2 (Humble)
Control Board	OpenRB-150, communicates with motors
Workspace	Flat table with ArUco markers, stable lighting

Fig. 1. Simple Hardware Setup

2.2 Control System / Algorithms

2.2.1 Camera Calibration

To enable accurate mapping from image coordinates to real-world positions, the camera was calibrated using a printed 10×7 checkerboard pattern. The checkerboard was fixed flat on a surface to avoid bending, and 20 images were captured from various angles and distances. OpenCV's findChessboardCorners() function[6], was used to detect the pixel positions of the inner corners of the checkerboard in each image. For each image, two sets of points were generated: 2D image points (pixel coordinates) and 3D object points, assuming the checkerboard lies on a flat surface (Z = 0). These point correspondences were passed to OpenCV's calibrateCamera() function, which computes the intrinsic parameters of the camera, including the camera matrix:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f_x and f_y are the focal lengths in pixels, and cx, cy are the coordinates of the optical center.

To correct lens distortion, five distortion coefficients $[k_1, k_2, p_1, p_2, k_3]$ were obtained from the calibration process. T These values help correct two types of distortion: radial distortion, which makes straight lines look curved (especially near the edges), and tangential distortion, which happens when the lens is not perfectly lined up with the image sensor[7]. The corrected pixel coordinates ($x_{\text{distorted}}, y_{\text{distorted}}$) are computed from the ideal normalized coordinates (x, y) using a standard distortion model:

$$x_{\text{distorted}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2)$$
$$y_{\text{distorted}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy$$

where $r^2 = x^2 + y^2$. These corrections ensure accurate image-to-world coordinate mapping in later stages of the project.

2.2.2 Detection and Tracking

After calibration, the camera continuously captured images of the workspace to detect ArUco markers in real time. OpenCV's ArUco module was used to detect each marker's unique ID and its four corner points in the image frame[8]. The marker center was computed by averaging the corner coordinates. A bounding box was drawn around each detected marker, and its ID along with its corresponding real-world coordinates (after transformation) was overlaid on the video feed. To account for a known spatial offset between the camera's origin and the robot's first joint, the marker located at the origin was assigned a base real-world coordinate of (-30, 0) mm. This provided a consistent reference frame for all subsequent transformations and robotic movement.

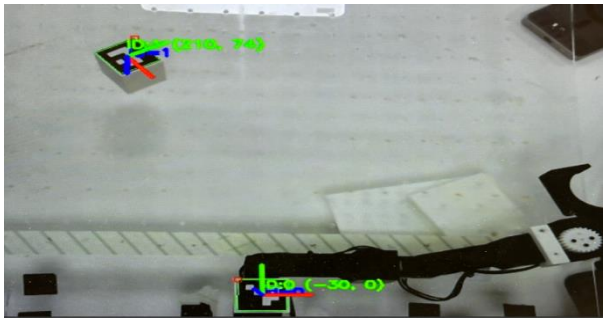

Fig. 2. ArUco Detection

2.2.3 Coordinate Transformation

I found that just using a simple mapping between pixel coordinates and real-world positions gave large errors. Because of that, the robot arm could not move accurately and failed to do the task properly. So, I used a regression model based on machine learning to build a better mapping. The camera and robot were fixed in place, and I placed an ArUco marker at 52 known points in the workspace. For each point, I recorded two sets of data: the pixel coordinates from the camera (x_{img}, y_{img}) and the real-world coordinates measured manually $(x_{\text{real}}, y_{\text{real}})$. A second-degree polynomial regression model was constructed using the following feature vector[9]:

$$\mathbf{X} = [1, x_{img}, y_{img}, x_{img}^2, x_{img} \cdot y_{img}, y_{img}^2]$$

And the conversion is expressed as:

$$\text{Real}_x = \sum_{i=0}^{2} \sum_{j=0}^{2-i} a_{ij} \cdot x^i y^j, \text{Real}_y = \sum_{i=0}^{2} \sum_{j=0}^{2-i} b_{ij} \cdot x^i y^j$$

Where are the pixel coordinates, and are learned coefficients for X and respectively. The converted coordinates are published as ROS topics and consumed by downstream nodes.

Separate regression models were trained for x_{real} and y_{real} . During execution, when a marker is detected, its pixel coordinates are used as input to the model to predict its real-world position. This transformation enables accurate and smooth robotic manipulation of objects based on camera feedback. Detailed data is provided in the Results section.

2.2.4 Gripper (End-Effector)

During the development of the end-effector, multiple gripper designs were created, built, and tested to determine the most suitable one for the task. The first prototype was too long and bulky, which limited the robot's workspace and made object manipulation difficult. The second version had high internal friction, requiring over 1000 mA of current to operate, which affected the overall system stability. Based on these observations, a third version was

designed and optimized to be more compact, lightweight, and energy-efficient. This final version was selected for integration into the system due to its improved functionality and better compatibility with the robotic arm.

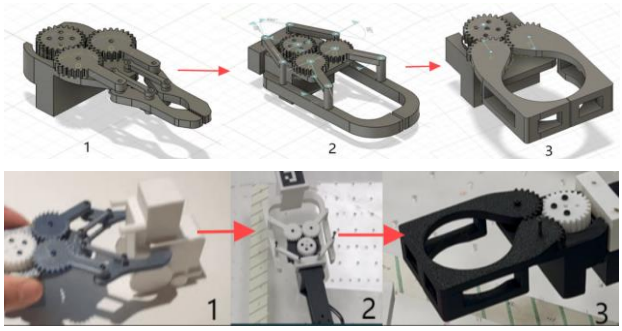

Fig. 3. Gripper Cad and Physical Prototype

In the gripper control system, we set 300 mA as the starting current to close the gripper normally, and 600 mA as the overcurrent threshold. When the current goes over 600 mA (I_{real}), the system assumes that the gripper has grabbed something and stops applying more force. This prevents the gripper from keeping too much pressure when it's empty, which could damage the motor. It also makes the system safer and more stable during real use.

$$I_{\text{real}} = |I_{\text{raw}}| \times 2.69$$

* 2.69 is the current scaling factor for the Dynamixel XC330 motor[10].

2.2.5 Inverse Kinematics

We tested several inverse kinematics (IK) methods to calculate the joint angles based on the end-effector's target position. We compared the Jacobian method[11], the Damped Least Squares (DLS) method[12], and an analytical method[13,14]. During testing, we found that both the Jacobian and DLS methods sometimes caused unstable or jerky movement, especially when the robot had to move quickly or change directions often. In contrast, the analytical method does not rely on iteration, runs faster, and gives more stable results, which makes it a better choice for our simple 2-DOF planar robotic arm.

So, we finally chose the analytical inverse kinematics method, implemented as the function anal_IK(). The formulas used are:

$$D = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}, \theta_2 = \arctan\ 2\left(\sqrt{1 - D^2}, D\right)$$

$$\theta_1 = \arctan\ 2(y, x) - \arctan\ 2(l_2 \cdot \sin\ \theta_2, l_1 + l_2 \cdot \cos\ \theta_2)$$

*l_1 : the length of the first link, l_2 : the length of the second link.

The calculated angles are then converted to degrees and sent to the motor control node. In real tests, this method gave us smoother movements, faster reaction, and better accuracy. That's why we used this approach in our final system.

2.2.6 Motion Strategy and Task Execution

To ensure safe and reliable operation, the robot system follows a structured task flow (with a total of 12 published and subscribed topics) combined with motion strategies that reduce errors and protect the object being handled. The overall process is managed by the commander node, which switches between task stages such as GRAB, AVOID, and PLACE. Object and obstacle positions are detected using the aruco_detector node and saved to local files (markers_positions and obstacle_positions). Based on the current stage, the arm_target_reader node reads the right target position and publishes it to the arm_target_pos topic.

When moving toward the object, the gripper is kept closed by default. This helps avoid unwanted contact or damage to the object before the robot has properly aligned with it. Once the arm gets close to the target, the system enables tracking mode for a second round of detection. This re-localizes the ArUco marker and allows the robot to make small adjustments for a more accurate approach before grabbing.

The overall workflow of the robotic system is illustrated in the flowchart below, which outlines the key stages. (The assembled robotic arm is shown in Appendix Figure B.)

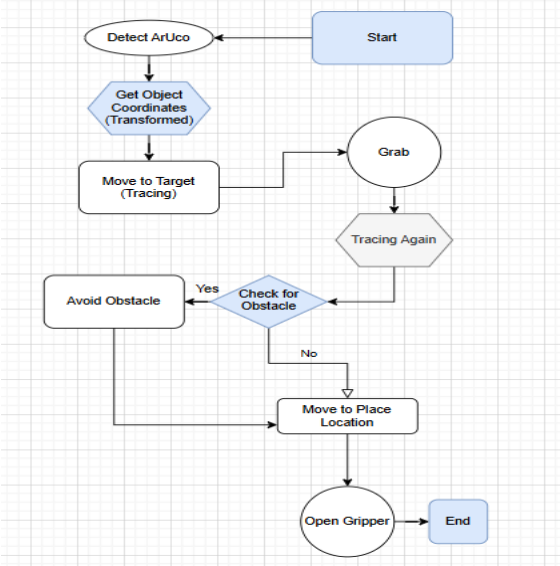

Fig. 4. Process Flowchart

2.2.7 Obstacle Avoidance

During obstacle avoidance and when moving to the place position, the system uses interpolated motion, meaning it won't go to the final position in a single jump. This makes the motion much smoother and helps avoid sudden turns or joint jolts that could cause the gripper to shake or drop the object. Especially for lightweight setups or delicate items, this approach helps improve safety and movement quality.

The robot's obstacle avoidance strategy is based on real-time vision feedback and local trajectory planning[15,16]. When an obstacle is detected, the system calculates its real-world center coordinate (x_o, y_o) and defines a hazardous area using a safety radius of 50 mm[17].

To determine if the robot is entering this region, the system computes the Euclidean distance from the robot's current position (x_r, y_r) to the obstacle center:

$$d = \sqrt{(x_r - x_o)^2 + (y_r - y_o)^2}$$

If $d < r_{\text{safe}}$, the robot switches to avoidance mode. Instead of crossing through the hazard zone, the robot follows points that lie near the boundary. These points can be estimated along a circular path using:

$$(x_i, y_i) = (x_o + r \cdot \cos\ \theta_i, y_o + r \cdot \sin\ \theta_i)$$

* θ_i represents evenly spaced angles along the obstacle perimeter.

To avoid sudden turns or jerky motion, the robot applies linear interpolation between each segment:

$$x_k = x_1 + \frac{k}{n+1}(x_2 - x_1), y_k = y_1 + \frac{k}{n+1}(y_2 - y_1)$$

2.2.8 GUI

The system is accessed and managed through MobaXterm, which provides an easy-to-use interface for remote control and monitoring.

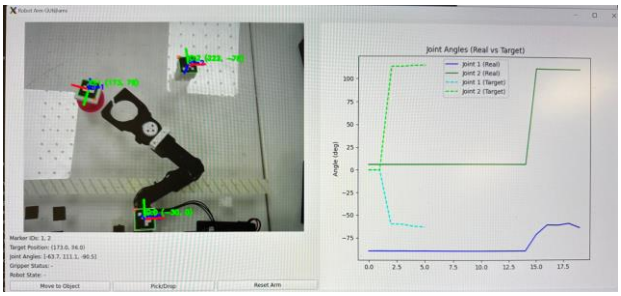

Fig. 5. GUI

2.3 Experimental Protocols

To evaluate the performance of the robotic system, we designed a structured set of experiments covering two key aspects: Accuracy of pixel-to-world coordinate transformation, and grasping and positioning performance in real robotic tasks. This section outlines how the data was collected and how the analysis was conducted. Results and evaluation will be presented in the next section.

2.3.1 Coordinate Transformation Accuracy

To validate the regression model used to convert camera pixel coordinates into real-world positions, we manually placed an ArUco marker at 52 known reference points within the workspace.

These data were used to train and test a second-degree polynomial regression model. The predicted real-world positions were compared with the ground-truth values to calculate the prediction error. Errors in both X and Y directions were recorded, and bar plots were later used to visualize the error distribution.

$$Errors = \sqrt{\{(x_a - x_t)^2 + (y_a - y_t)^2\}}$$

2.3.2 Comparison Between Single and Double Detection

To evaluate how the number of tracking steps affects positioning accuracy before grasping, we designed a simple comparison test between single detection and double detection. In the single detection setup, the robot moves close to the object and performs one round of tracking before grasping. In the double detection setup, after the first tracking and approach, the robot performs a second tracking step to refine its alignment before executing the grasp.

We selected five fixed target points within the workspace. At each point, we ran 5 trials for both single and double detection, resulting in a total of 50 trials. For each trial, we recorded the pre-grasping error, defined as the distance between the robot's end-effector and the target point just before grasping. All tests were done under consistent lighting and conditions to ensure fair comparison. The goal of this test was to see whether adding a second detection step could noticeably improve the positioning accuracy before grasping.

2.3.3 Grasping Accuracy and Repeatability

To assess the grasping stability and repeatability, the robot was instructed to grasp a target object at five different positions, each repeated ten times. The final position of the gripper tip was recorded during contact, and the deviation from the intended target position was calculated.

This data was used to analyze how different workspace regions (e.g., near singularities or with different inertia) affect grasping precision. No external disturbances were applied, and the robot followed the same motion plan each time.

2.3.4 Data Collection and Analysis

Positional errors were calculated using the Euclidean distance between the predicted and actual coordinates. For each group of trials, both the average and maximum errors were measured to evaluate positioning accuracy. Grasp success was determined based on a combination of current sensor readings and manual observation. For the obstacle avoidance tests, success was judged visually — as long as the robot managed to avoid the obstacle and complete the task without interruption, the attempt was considered successful. All results were later visualized in the Results section using graphs and scatter plots to highlight overall trends and compare performance across different conditions.

3. RESULTS

This section presents the evaluation results based on the experimental protocols described earlier. The focus is on the accuracy of coordinate transformation, the effect of detection strategies on grasp positioning, grasping repeatability, and obstacle avoidance success.

3.1 Coordinate Transformation Accuracy

To evaluate the accuracy of pixel-to-world coordinate mapping, we tested a trained second-degree regression model using 52 manually labeled reference points across the workspace. The results showed an average prediction error of 2.51 mm in the X-axis and 5.38 mm in the Y-axis, with slightly larger Y errors concentrated near the top-left and bottom-right corners. The maximum observed Y-axis error was 8.4 mm, while the smallest X-axis error was only 0.9 mm. Over 90% of all test points had errors below 6 mm, demonstrating that the mapping model is accurate enough for reliable robotic grasping.

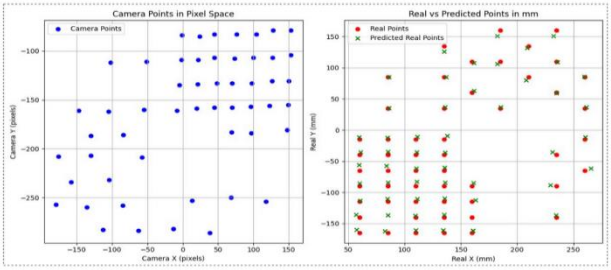

Fig. 6. Coordinate Transformation

*Left: Detected pixel coordinates from ArUco markers. Right: Comparison of predicted vs. true real-world positions using the trained regression model.

(Videos Link: Advanced Medical Robotics)

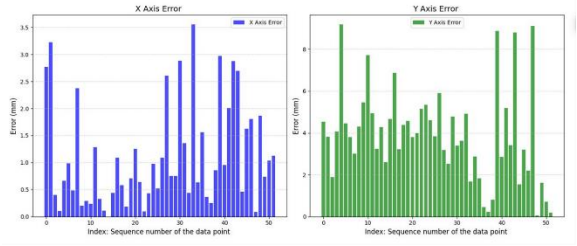

Fig. 7. Error Plot

*Left: X-axis prediction errors for each of the 52 test points.

Right: Y-axis prediction errors for the same points.

The error is calculated as the difference between the predicted and actual real-world coordinates.

3.2 Pre-Grasping Accuracy

We compared the effectiveness of single detection and double detection strategies by performing five trials for each method at five fixed target points. The average pre-grasping error for single detection was 14.92 mm, whereas double detection reduced this to 5.28 mm. The maximum errors recorded were 17.8 mm for single detection and 6.4 mm for double detection. Additionally, double detection showed lower standard deviation (1.9 mm compared to 4.5 mm), indicating more stable performance. As shown in Figure 3, grasping points obtained from single detection (red) were more scattered, while those from double detection (blue) formed tighter clusters closer to the target positions.

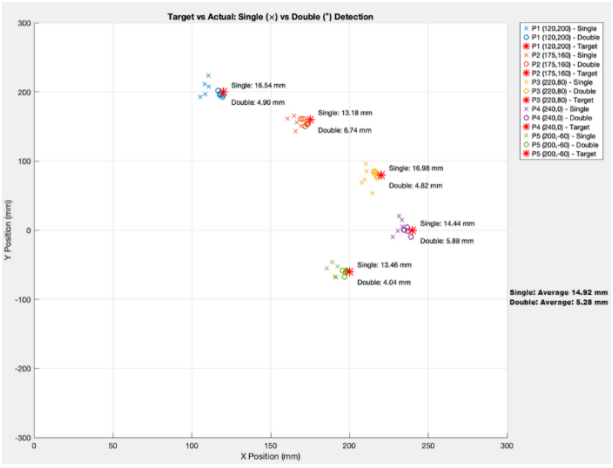

Fig. 8. Single Detection and Double Detection

3.3 Grasping Accuracy and Repeatability

To assess the robot's repeatability, we performed 10 grasping attempts at each of five predefined locations. The green zone, near the center of the workspace, showed the best consistency, with all errors below 4 mm, while the purple zone, located near a kinematic singularity, produced the worst performance, with an average error above 10 mm and a maximum of 14 mm. The other three locations showed average deviations below 6 mm. As visualized in Figure 5, grasping points in the green zone formed tight clusters, whereas those in the purple zone were more widely dispersed and offset, indicating that repeatability is influenced by the robot's pose and stability within the workspace.

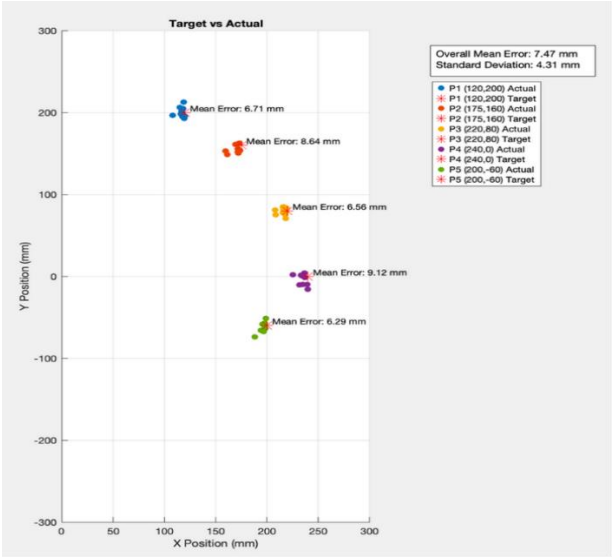

Fig. 9. Target and Actual Errors

3.4 Motion Path and Workspace Visualization

Figure 10 illustrates the robot's full pick-and-place motion with integrated obstacle avoidance and workspace constraints. The leftmost plot shows the interpolated trajectory from the initial position (0.00, -280.00 mm) to Target 1 (236.34, 70.95 mm), with a smooth segmented path designed to avoid abrupt joint movements. After reaching Target 1, the robot executes a second motion toward Target 2 (203.79, -157.71 mm), shown in the middle plot, which also follows an interpolated path to ensure motion continuity.

The final plot on the right provides a complete visualization of the robot's reachable workspace

when θ_2 (elbow joint) is constrained between 0° and 90°, forming a green crescent-shaped feasible area. Both Target 1 and Target 2 lie well within this zone, confirming that the chosen inverse kinematics solutions maintain the robot in a stable and physically reachable configuration throughout the motion.

The trajectory was computed using analytical inverse kinematics, with link lengths of 100 mm each and an 80 mm end-effector. Joint angles after reaching Target 1 were $\theta_1 = 67.97°$ and $\theta_2 = 59.09°$, while after reaching Target 2, $\theta_1 = -69.11°$ and $\theta_2 = -10.91°$, indicating symmetric but distinct configurations. This not only validates the interpolation strategy and obstacle-aware path selection but also reflects the system's ability to transition smoothly between goals without violating kinematic constraints or entering hazardous zones.

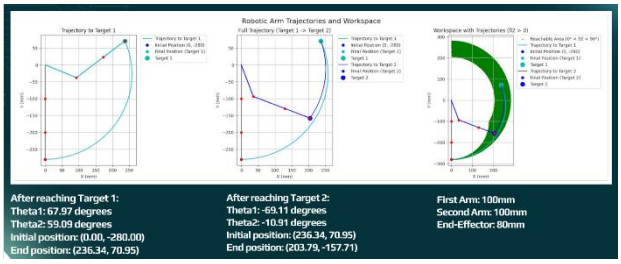

Fig. 10. Trajectories and Workspace

3.5 Obstacle Avoidance Evaluation

When an obstacle was detected, the robot initiated avoidance mode and planned a new path based on the defined hazardous area. Each test involved the robot moving toward a goal while dynamically rerouting around the obstacle. The success of each avoidance attempt was determined through visual observation: if the robot successfully bypassed the obstacle and reached the goal without collisions or unexpected stops, the trial was counted as successful. In all test cases, the robot was able to complete the task without entering the hazard zone or deviating from the avoidance logic. The motion was smooth and stable throughout, with no manual intervention required. A video demonstration of the avoidance behavior is available for reference.

(Include all experiments videos: Advanced Medical Robotics)

3.6 Risk Assessment

To keep the system safe and reliable, we did a basic risk review of each part of the process. The table lists the main things that could go wrong—from failing to grab objects, hitting obstacles, missing markers, to full software crashes. Each row shows what could cause the problem, how serious it is, how often it might happen, and what we can do to prevent it.

For example, RISK-01 is about the gripper failing to hold an object, maybe due to weak force or bad alignment. RISK-02 shows how the arm might hit something if the inverse kinematics or path plan is off. The most critical one is RISK-08: if the ROS node suddenly crashes, the whole robot could stop mid-task—that's the highest-scoring risk (20).

We also added suggestions for each case. Things like adding current checks to confirm grip, giving extra buffer space when planning moves, improving lighting to help the camera, or adding emergency stop commands. These steps help a lot during testing, but in a real medical setting, they'll need more work.

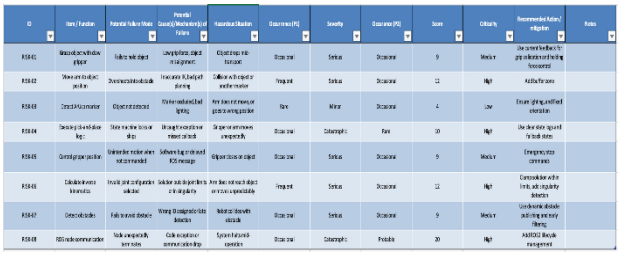

Fig. 11. Trajectories and Workspace

4. DISCUSSION

This section interprets the results presented earlier, discusses their implications for the design and use of vision-guided robotic systems, and connects the findings to broader trends and needs in the field of medical robotics.

4.1 Mapping Accuracy

The accuracy of pixel-to-world coordinate transformation was a key factor in ensuring the robot could move precisely. Based on 52 labeled data points, the regression model achieved acceptable error margins, with X-axis predictions showing

slightly better consistency than Y-axis predictions. Some outliers appeared near the workspace edges, but overall, the model provided dependable input for motion planning.

These results are especially important in environments where space is constrained and mistakes can't be tolerated—such as around patients or lab equipment. We also visualized the full reachable area of the robot under joint limits, confirming that the grasping tasks were performed well within this zone. The robot's effective workspace, shaped by link lengths and joint ranges, acted as a useful guide during trajectory design. Staying within that area ensured smooth, singularity-free movements during all tasks.

4.2 Detection Strategy

One of the clearest findings was how much a second round of visual detection (tracking) improved performance. When the robot used just one detection to approach and grasp, the average pre-grasp error was nearly 15 mm. Adding a second detection step just before grabbing cut that down to around 5 mm. Not only did the accuracy improve, but the grasp attempts were also more consistent, with fewer outliers.

This is particularly relevant in real-life settings where objects may shift slightly or where lighting might fluctuate. The extra detection essentially gave the robot a chance to double-check before acting. For use in clinical or lab settings—where grabbing the wrong vial or tool could cause problems—this added layer of reliability is worth the extra few seconds it takes.

4.3 Grasping Consistency

The robot's ability to repeat the same motion accurately depends heavily on where in the workspace the task is performed. When the robot operated closer to its central region (green zone), grasping errors stayed under 4 mm. However, tasks done near the edges (especially near kinematic singularities) produced wider spreads, with some errors going up to 14 mm.

The robot was less stable near these critical points—small angle changes resulted in large end-effector shifts. This matters a lot when designing workflows in hospital rooms or clinics, where the robot may need to interact with patients or instruments placed at specific locations. Being aware of these pose-related variations helps ensure tasks are executed in more stable regions.

4.4 Obstacle Avoidance and Workspace Feasibility

Obstacle avoidance was handled using a simple but effective strategy. The robot monitored for nearby ArUco markers that represented obstacles, then calculated a safety buffer around them. It would follow the edge of this buffer zone instead of trying to go through it. This behavior was consistent and worked in all tests without failure.

We also applied interpolated motion throughout both regular and avoidance paths. Rather than jumping directly from point A to point B, the robot broke the motion into smaller segments. This made the arm's movement much smoother and avoided sudden turns that could shake the object or damage the gripper. The full trajectories to two example targets were visualized against the robot's reachable area, and all joint angles stayed within safe limits throughout. After reaching each target, the calculated joint positions (e.g., $\theta_1 = 67.97° / –69.11°$) confirmed that the robot stayed well within its mechanical range, and the final locations fell inside the validated workspace map.

4.5 Implications for Medical Robotics

From a broader perspective, this system shows how relatively simple and affordable components—like ArUco-based vision and basic 2-DOF arms—can still deliver reliable, intelligent behavior when well-integrated. The techniques used here, especially the double detection strategy and path smoothing, are highly applicable to medical scenarios[18].

Healthcare settings often involve repetitive but delicate tasks, such as sample handling, assisting with personal items, or preparing instruments. These don't necessarily require high-end robots, but they

do demand precision and safety. The system developed in this project could be adapted for bedside assistance, lab automation, or even sterile environments like operating rooms, especially where reducing human contact is critical.

4.6 Limitations and Future Plan

One of the most fundamental limitations of the system lies in its mechanical structure: it operates with only two degrees of freedom, and motion is constrained to a 2D plane (X-Y). While this simplifies control and reduces hardware cost, it severely limits the robot's flexibility. The arm cannot reach over or around objects, cannot interact with items at different heights, and is unable to adapt to situations requiring vertical motion or tool angling. Additionally, current-based detection is binary and lacks adaptive pressure feedback, which limits the ability to safely grasp fragile or deformable objects. In real medical environments—such as hospital rooms or clinics—space is often three-dimensional and irregular. For example, a patient's bedside may include equipment at various heights, or items may be placed on elevated surfaces. The current robot would not be able to access such areas without adding a vertical (Z-axis) component or at least a third joint.

The second major constraint comes from the system's performance in dynamic scenes. The obstacle detection logic is based entirely on static visual input from a fixed-position camera. This approach works well for known, static obstacles with clear markers but is not suited for unpredictable environments[19]. The robot cannot detect or respond to moving obstacles, such as people walking by, swinging IV poles, or objects falling. Furthermore, if lighting conditions change or objects are partially occluded, the current vision algorithm may fail to detect the target or obstacle entirely. Compounding this limitation, the current obstacle avoidance algorithm only handles one obstacle at a time; additional obstacles are not processed simultaneously, which may cause collisions in more complex environments

Lastly, the system relies exclusively on ArUco markers for both object and obstacle identification. This method is practical in lab settings, but unrealistic in real-life use. Medical staff will not place printed markers on every tool, bottle, or surface. Without a marker, the system has no ability to recognize or classify what it sees. This fundamentally limits its generalizability and requires significant redesign in the perception pipeline if the system were to be deployed outside of tightly controlled environments.

To address these issues, several improvements are possible. First, replacing the fixed camera with a mobile or wrist-mounted stereo camera would allow for real-time depth perception and dynamic scene understanding. Using onboard sensing like LiDAR could also allow for marker-free obstacle detection. The perception pipeline could be expanded to include object detection models trained on real medical tools or patient-area items. Upgrading to a 3-DOF manipulator for enhanced dexterity and extending the obstacle avoidance algorithm to handle multiple dynamic objects simultaneously.

Despite these limitations, the project demonstrated that reliable and adaptable robotic automation is possible using modest hardware, thoughtful planning, and a clear control architecture. The success of the system in structured conditions lays a solid foundation for future extensions into real-world medical environments.

REFERENCES

[1] M. T. Thai, P. T. Phan, S. Wong, N. H. Lovell, and T. N. Do, 'Advanced Intelligent Systems for Surgical Robotics', Jan. 02, 2020, arXiv: arXiv:2001.00285. doi: 10.48550/arXiv.2001.00285.

[2] D. Piaggio et al., 'The use of smart environments and robots for infection prevention control: A systematic literature review', American Journal of Infection Control, vol. 51, no. 10, pp. 1175–

1181, Oct. 2023, doi: 10.1016/j.ajic.2023.03.005.

[3] K. Thurow, ‘Strategies for automating analytical and bioanalytical laboratories’, Anal Bioanal Chem, vol. 415, no. 21, pp. 5057–5066, Sep. 2023, doi: 10.1007/s00216-023-04727-2.

[4] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, ‘Automatic generation and detection of highly reliable fiducial markers under occlusion’, Pattern Recognition, vol. 47, no. 6, pp. 2280–2292, Jun. 2014, doi: 10.1016/j.patcog.2014.01.005.

[5] S. Hutchinson, G. D. Hager, and P. I. Corke, ‘A tutorial on visual servo control’, IEEE Trans. Robot. Automat., vol. 12, no. 5, pp. 651–670, Oct. 1996, doi: 10.1109/70.538972.

[6] ‘OpenCV Documentation, calibrateCamera() - Camera Calibration and 3D Reconstruction.’ [Online]. Available: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#gaedde880b30c8a2e34c6661c6f5f65974.

[7] Z. Zhang, ‘A flexible new technique for camera calibration’, IEEE Trans. Pattern Anal. Machine Intell., vol. 22, no. 11, pp. 1330–1334, Nov. 2000, doi: 10.1109/34.888718.

[8] ‘Tutorial_aruco_detection.html’. [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html

[9] T. Hastie, R. Tibshirani, and J. H. Friedman, The elements of statistical learning: data mining, inference, and prediction, Second edition. in Springer Series in Statistics. New York, NY: Springer, 2017.

[10] ‘DYNAMIXEL XC330 User Manual’. [Online]. Available: https://emanual.robotis.com/docs/en/dxl/x/xl330-m288/

[11] K. Tchon, ‘Optimal Extended Jacobian Inverse Kinematics Algorithms for Robotic Manipulators’, IEEE Trans. Robot., vol. 24, no. 6, pp. 1440–1445, Dec. 2008, doi: 10.1109/TRO.2008.2006240.

[12] A. S. Deo and I. D. Walker, ‘Overview of damped least-squares methods for inverse kinematics of robot manipulators’, J Intell Robot Syst, vol. 14, no. 1, pp. 43–68, Sep. 1995, doi: 10.1007/BF01254007.

[13] J. J. Craig, Introduction to robotics: mechanics and control, Third edition, Indian subcontinent adaptation. Chennai: Pearson, 2009.

[14] M. Wilson, Ed., ‘Robot Modeling and Control’, Industrial Robot: An International Journal, vol. 33, no. 5, pp. 403–403, Sep. 2006, doi: 10.1108/ir.2006.33.5.403.1.

[15] J. Park, J.-H. Lee, and S. H. Son, ‘A Survey of Obstacle Detection Using Vision Sensor for Autonomous Vehicles’, in 2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), Daegu, South Korea: IEEE, Aug. 2016, pp. 264–264. doi: 10.1109/RTCSA.2016.54.

[16] R. Singh, T. K. Bera, and N. Chatti, ‘A real-time obstacle avoidance and path tracking strategy for a mobile robot using machine-learning and vision-based approach’, SIMULATION, vol. 98, no. 9, pp. 789–805, Sep. 2022, doi: 10.1177/00375497221091592.

[17] L. D. Hanh and V. D. Cong, ‘Path Following and Avoiding Obstacle for Mobile Robot Under Dynamic Environments Using Reinforcement Learning’, Journal of Robotics and Control, vol. 4, no. 2, pp. 157–164, Apr. 2023, doi: 10.18196/jrc.v4i2.17368.

[18] G.-Z. Yang et al., ‘Medical robotics—Regulatory, ethical, and legal considerations for increasing levels of autonomy’, Sci. Robot., vol. 2, no. 4, p. eaam8638, Mar. 2017, doi: 10.1126/scirobotics.aam8638.

[19] D. Kragic and M. Vincze, ‘Vision for Robotics’, FNT in Robotics, vol. 1, no. 1, pp. 1–78, 2009, doi: 10.1561/2300000001.

APPENDIX

Category	Component / System	Description
Mechanical	**2× Dynamixel XC330-M288-T (for robotic arm)**	Control the base and shoulder joints of the robotic arm
	1× Dynamixel XC330-M288-T (for gripper)	Controls the open/close movement of the custom-designed gripper
	3D-printed gripper (multiple versions)	Custom-designed and optimized; final version is lightweight and stable
Control Unit	OpenRB-150 Controller Board	Interfaces between all motors and Raspberry Pi; handles serial communication and current sensing
	Raspberry Pi 4	Main controller; runs ROS2 and computer vision code
Perception	USB Camera	Mounted above the workspace; detects ArUco markers and objects
Software	Raspberry Pi OS (Linux)	Operating system for the control board
	ROS2 Humble	Robot Operating System framework for motion and communication
	OpenCV	Used for image capture, camera calibration, and ArUco marker detection
	Python + Polynomial Regression	Used to map pixel coordinates to real-world positions
Workspace	Flat table + ArUco markers	Defines the working area; markers enable accurate coordinate transformation
	Unified power supply	Provides stable power and avoids cable interference during movement

Figure.A. Hardware Setup

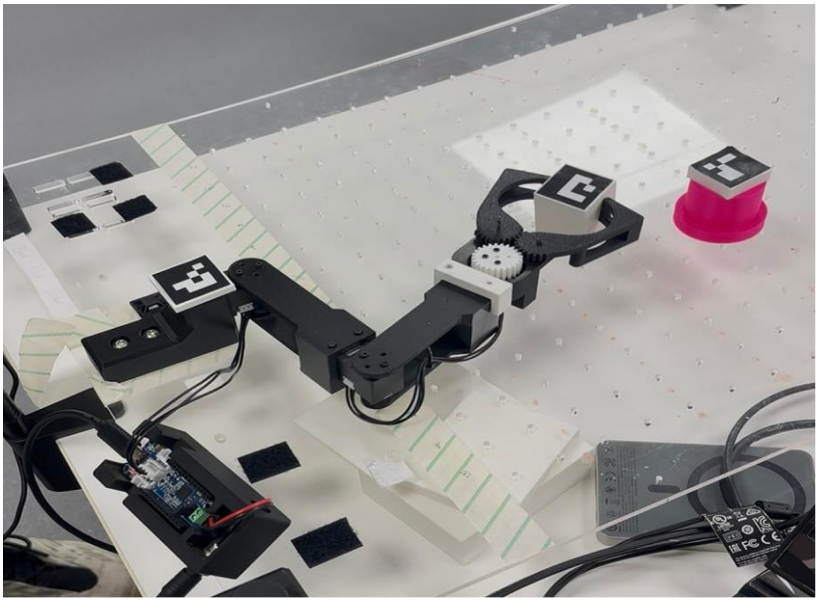

Figure.B. The Assembled Robotic Arm