Obstacle Avoidance Pick and Place on a UR5e Robot: A MATLAB Simulation

1st Aditya Nisal
Robotics department
WPI
anisal@wpi.edu

2nd Merna Elbayoumi
Robotics department
WPI
melbayoumi@wpi.edu

3rd Sri Lakshmi Hasitha
Robotics department
WPI
sbachimanchi@wpi.edu

4th Tanish Mishra
Robotics department
WPI
tamishra@wpi.edu

***Abstract*—This project aimed to devise and validate a comprehensive solution for an obstacle avoidance pick and place task using a UR5e robot with a Robotiq 2F85 gripper attachment. Given the constraints in accessing the physical robot at the Washburn lab, we leveraged the power of simulation using MATLAB to validate our approach. The core of our solution involved implementing the Rapidly Exploring Random Tree (RRT) algorithm for efficient path planning and motion control. The simulation demonstrated successful execution of the pick and place task while avoiding obstacles. This outcome effectively highlights the robustness of our approach and the potential of the RRT algorithm for sophisticated robot manipulations.**

***Index Terms*—UR-5e robot arm, Obstacle avoidance, Pick and place, Sensory system, Rapidly Exploring Random Tree, MATLAB**

I. Introduction

In the domain of robotics, pick and place operations are common yet crucial tasks, particularly in industries such as manufacturing, warehousing, and logistics[9,14]. However, these operations often involve challenges, such as navigating through complex environments and avoiding obstacles, which require sophisticated planning and control strategies[9,11]. This project aimed to address these challenges and devise an efficient solution for an obstacle avoidance pick and place task using the UR5e robot[15].

The UR5e is a versatile robot arm from Universal Robots, known for its flexibility, accuracy, and ease of programming[1]. With its 5 kg payload capacity, 850 mm reach, and repeatability of ±0.03 mm, it is well-suited for a wide range of tasks, including pick and place operations[1,14]. In our project, the UR5e was equipped with a Robotiq 2F85 gripper, a widely used adaptive gripper known for its precise control and robust design[15].

Initially, our project was planned to be implemented on the physical UR5e robot housed in the Washburn lab[3]. However, due to limited access to the lab and the robot - as it was shared among multiple students - we decided to pivot our approach[7]. We turned to simulation as a viable alternative, given its flexibility and the ability to iterate quickly without the risk of damaging the physical equipment[3,7].

For our simulation, we selected MATLAB, a powerful computation tool offering a robust environment for developing and visualizing robot manipulations[16]. MATLAB's Robotics System Toolbox provided the necessary functionality to represent the UR5e robot and the Robotiq gripper, to define the environment, and to implement the path planning and control algorithms[16].

The cornerstone of our solution was the Rapidly Exploring Random Tree (RRT) algorithm, a popular method for path planning in robotics[10,12]. RRT is particularly suited for high-dimensional configuration spaces and for navigating through complex geometrical constraints[10,13]. By generating a tree of possible paths that rapidly spans the configuration space, RRT provides an efficient way to find feasible paths in environments with obstacles[10,13].

This report will detail the methods we adopted, the challenges we encountered, and the results we obtained through our MATLAB simulation. Our hope is that this work will contribute to the broader understanding of implementing complex robotic tasks using simulated environments and advanced path planning algorithms like RRT[12,14,15].

II. Methods

2.1.0 Methodology Adapted for real robot

In the actual implementation on the UR5e robot, we performed inverse kinematics - a fundamental concept in robotics that involves calculating the joint parameters needed to place the end-effector of a robot in a desired position and orientation. The primary inputs for the inverse kinematics process were the current joint angles and the transformation matrix of the end-effector. The objective was to compute the joint angles that correspond to the given transformation matrix.

The process of inverse kinematics is not straightforward because there's no simple mathematical equation to directly calculate the joint angles from the end-effector's position and orientation. Instead, we used the Newton-Raphson method, a widely used numerical method for root finding, to solve this problem.

The approach involved setting up the problem as the root-finding equation:

$$\Theta_{\text{bar}} = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6] \tag{1}$$

$$f(\Theta_{\text{bar}}) - T_d = 0 \tag{2}$$

where Θ_{bar} represents the set of joint angles $[\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$, function f represents the forward kinematics function that computes the end-effector's transformation matrix for a given set of joint angles, and T_d is the desired transformation matrix of the end-effector.

The Newton-Raphson method was then used to iteratively update the joint angles until the computed transformation matrix closely matched the desired transformation matrix. The initial guess for the Newton-Raphson method was the current joint angles of the robot.

Once the desired joint angles were obtained, we employed a velocity controller to smoothly move the robot from its initial position to the desired position. The velocity controller ensured that the robot transitioned between these positions at a controlled speed, thus preventing any abrupt or fast movements that could potentially harm the robot or its surroundings.

2.2.0 Methodology Adapted for simulation

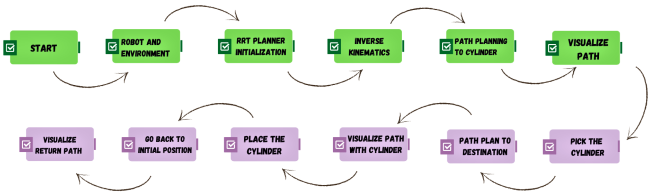

Fig. 1. Methodology Adapted in the simulation

2.2.1 Start:

In the simulation's initial stage, MATLAB's 'clear' command was invoked to ensure a clean workspace. This step was necessary to avoid any potential interference from residual variables or scripts from previous operations.

2.2.2 Robot and Environment:

Next, the UR5e robot model was loaded using MATLAB's 'loadrobot' function, and its home configuration was adjusted accordingly. The end effector of the UR5e was then replaced with the Robotiq 2F85 gripper. The environment was then created, comprising a bench, a belt, a barricade, and a cylindrical object represented as collision bodies. This created an environment in which the robot could interact, providing a realistic simulation for the robot's operations.

2.2.3 RRT Planner and Initialization:

An RRT (Rapidly-exploring Random Tree) planner was set up with the robot and the defined environment. The RRT planner was chosen due to its effectiveness in high-dimensional spaces and its ability to handle non-linear constraints, making it ideal for robotic manipulation tasks. Key planner parameters such as MaxConnectionDistance, ValidationDistance, and the heuristic settings were tuned to optimize the robot's trajectory planning process.

2.2.4 Inverse Kinematics:

The first instance of inverse kinematics was solved using MATLAB's 'inverseKinematics' function. The goal was to calculate the joint parameters that would allow the robot's end effector to reach the picking location. This step was crucial for enabling the robot to move toward the cylindrical object accurately.

2.2.4 Path Planning to Cylinder:

The previously initialized RRT planner was used to plan a path from the robot's initial configuration to the goal configuration. The RRT planner's primary function here was to ensure a collision-free path from the robot's starting position to the cylindrical object's location.

2.2.5 Visualize Path:

MATLAB's 'show' function was used to visualize the planned path. This visualization provided a step-by-step depiction of the robot's movement from its initial location to the object, providing a powerful visual confirmation that the planning process was functioning as expected.

2.2.6 Pick the Cylinder:

In this step, the robot was instructed to pick up the cylindrical object. This was accomplished by closing the gripper, creating a new rigid body for the cylinder, and adding it to the robot's rigid body tree, effectively simulating the physical act of picking up the object.

2.2.7 Path Plan to Destination:

With the cylinder now attached, the robot's next task was to transport the object to a predetermined destination. Similar to the path planning to the cylinder, inverse kinematics was solved for the destination, and the RRT planner was used to generate a collision-free path from the current location to the destination.

2.2.8 Visualize Path with Cylinder:

The robot's path from the cylindrical object's location to the destination was visualized, showing the robot successfully navigating the environment while carrying the object.

2.2.9 Place the Cylinder:

Upon reaching the destination, the robot simulated the action of placing the cylinder. This involved opening the gripper, detaching the cylinder from the robot's body tree, and adding it back to the environment as a collision object.

2.2.10 Go Back to the Initial Position:

Now that the pick and place task was complete, the robot was directed to return to its initial position. Using the RRT planner, a path was generated that guided the robot back to its starting position, avoiding any obstacles along the way.

2.2.11 Visualize Return Path:

The final stage of the simulation involved visualizing the robot's return path. This served as confirmation that the robot had successfully completed its task and returned to its original position, marking the completion of the simulated operation.

2.3.0 Rapidly-exploring Random Tree used

Fig. 2. RRT used

2.3.1 Initialization of RRT:
The RRT algorithm starts with an initial node, typically the starting position of the robot. This node represents the root of the tree that the RRT algorithm will expand in the configuration space. In this project, the robot's initial configuration served as the root node.

2.3.2 Configuration Space Embedding:
The configuration space, often abbreviated as C-space, represents all possible configurations (or positions) of the robot. Each node in the tree has a position in this configuration space. As the tree expands, it populates the configuration space with nodes that represent feasible robot configurations, covering an increasing proportion of the available space.

2.3.3 Expansion of Tree:
The tree expansion in RRT is conducted iteratively. In each iteration, a random point is sampled in the configuration space. The algorithm then finds the closest node in the tree to this random point. A new node is then created in the direction of the random point from the closest node, subject to a maximum allowed distance. This new node is added to the tree, expanding its coverage of the configuration space.

2.3.4 Collision Checking:
Collision checking is a vital part of the RRT algorithm. Whenever a new node is added to the tree, a check is performed to ensure that the robot configuration represented by that node, and the path to reach that configuration, doesn't result in a collision with any of the obstacles present in the environment. In this project, the manipulatorRRT function in MATLAB takes care of the collision checking.

2.3.5 Path Generation:
Once the RRT has been sufficiently expanded, path generation begins. If the RRT has reached the goal configuration, a path can be traced from the goal back to the root of the tree (the starting configuration of the robot). This path represents a feasible sequence of robot configurations that connect the starting position to the goal position without colliding with any obstacles.

In the context of the project, the RRT algorithm was used twice: first to plan the path from the initial position to the cylinder, and then to plan the path from the cylinder to the desired destination. In both cases, the paths generated by the RRT algorithm were collision-free and connected the starting and goal configurations of the robot.

After generating the path can be further refined by smoothing or interpolating between the configurations to create a more visually appealing and efficient trajectory. In this project, the interpolate function was used to create a more continuous path for the robot to follow.

III. Challenges

3.1.0 Real UR5 Robot Challenges:
Working with a real UR5 robot presented its own unique set of challenges. One of the primary issues was limited access to the robot as it was being shared among multiple students in the Washburn lab. This not only limited the amount of time we had for testing and debugging but also made it difficult to maintain a consistent and controlled environment for our experiments.

The intricacies of physical hardware also made the task more complex. Minor variations in sensor readings, actuator responses, and even the robot's physical state (like battery level or wear and tear) could significantly impact the robot's performance. Also, safely operating the robot while ensuring it didn't harm its surroundings or itself was a constant concern.

3.2.0 Matlab Simulation Challenges:
While moving to a Matlab simulation overcame some of the difficulties of working with a real robot, it introduced new challenges.

3.2.1 Obstacle Representation: In the simulated environment, representing the obstacles accurately was a challenge. We had to ensure that the size, shape, and location of the obstacles in the simulation accurately reflected the real-world scenario. This required careful measurement and coding to ensure a faithful representation.

3.2.2 Choosing and Tuning the Path Planning Algorithm: The selection of an appropriate path planning algorithm is crucial for the success of the project. We chose the RRT algorithm due to its ability to handle high-dimensional spaces and its proficiency in navigating complex environments. However, tuning the parameters of the RRT algorithm to work effectively in our specific scenario required significant experimentation and adjustment.

3.2.3 The complexity of the Model: The UR5e robot is a complex system with multiple degrees of freedom. Accurately modeling this in Matlab, while also considering the added complexity of the robotiq2F85 gripper, made the task more complex. Understanding and implementing the kinematics and dynamics of such a system in a way that accurately reflected the real robot's behavior was a significant challenge. Moreover, the complexity of the model increased the computational demand of the simulation. This meant that the processing time

required for path planning, inverse kinematics calculations, and visualization could become substantial, especially when iterating and testing various scenarios. Balancing the level of detail in the model with computational efficiency was a key challenge in the development of the simulation.

IV. Results

The results of our project are twofold, involving both real-world scenarios and simulation of an obstacle-avoiding pick-and-place task using the UR5e robot.

In the real-world scenario, the UR5e robot performed a pick-and-place task with a pair of scissors using regular inverse kinematics. While this task did not involve obstacle avoidance, it served as an effective way to build an understanding of the communication protocols of the UR5e while also being a method of validation for the numerical inverse kinematics solution generated by our code.

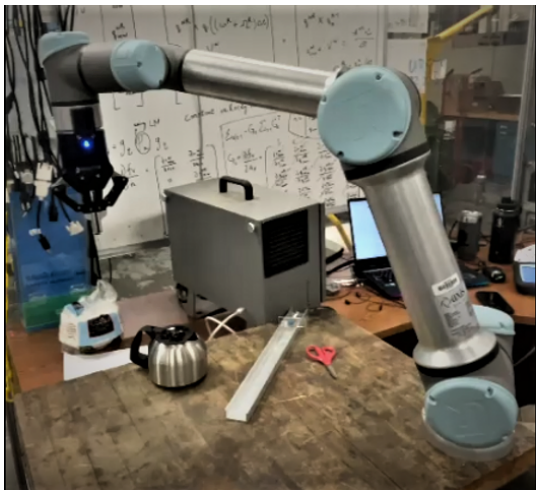

Fig. 3. The beast (UR-5e robot) at Washburn Labs.

We captured this operation in a video that shows the real robot arm executing the task. This demonstration underscores the precision and adaptability of the UR5e robot in a real-world task. The demonstration can be viewed in this `Video`

Following the real-world implementation, we turned to a simulated environment due to limited access to the physical robot. In the simulation, we designed a small wall to serve as an obstacle in front of the robot. The robot's task was to pick up a cylinder positioned on its side, navigate around the wall, and place the cylinder on the wall's opposite side. We provide two photos from the Matlab simulation illustrating the initial and final positions of the robot arm.

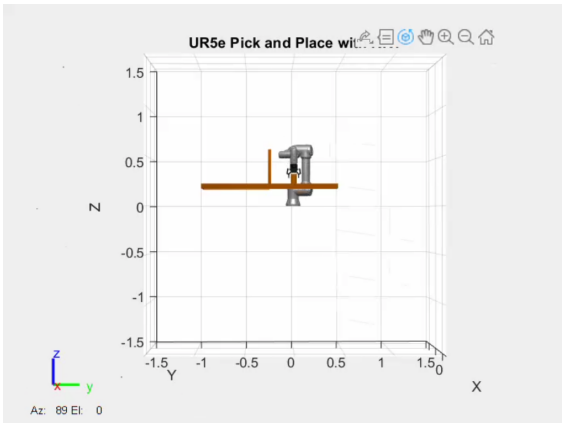

Fig. 4. Ready to pick

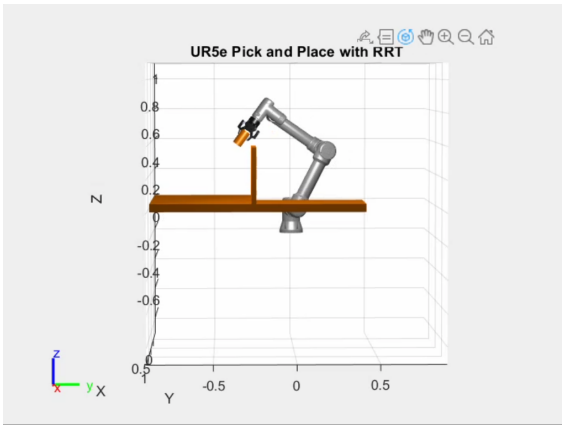

Fig. 5. Overcoming the obstacle

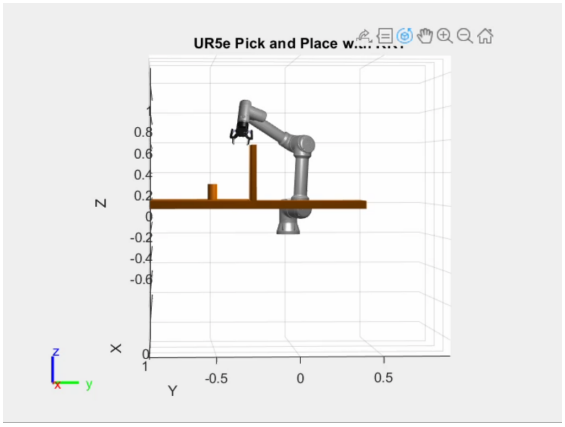

Fig. 6. Placed the cylinder

The initial photo shows the robot arm ready to pick up the cylinder, the second one shows the cylinder being picked by the robot and in the middle of the process, and the final position photo demonstrates the robot arm after successfully placing the cylinder at the desired destination, having avoided the wall. This successful completion of the task in the Matlab simulation is also demonstrated in a recorded `Video`.

The comparison between the real robot arm's movement and the simulation reinforces the accuracy of our model and control methodologies. Moreover, the differences between the real and simulated environments highlight the adaptability of our project approach in the face of limited access to the real robot.

V. Conclusion

The completion of this project led to several significant insights and lessons, shaping our understanding and approach to robotics and motion planning. One crucial lesson we learned is the importance of ensuring access to necessary resources, such as the UR5e robot, before committing to a project. Limited accessibility to the robot posed a significant challenge and necessitated a shift in our approach from real-world implementation to simulation.

While our project successfully achieved its goal, we recognize that there are several opportunities for future development and alternate methods that could have been explored. One such area for future work is the application of this project to a broader range of tasks in a real-world setting. This would involve overcoming the challenges we faced with the real robot, such as improving the reliability and efficiency of the robot's operation, by fine-tuning control algorithms and enhancing the robot's interactions with its environment.

Another area for future work involves expanding the simulation environment. While we used Matlab for our project, other simulation software like Gazebo could be considered for more complex and realistic simulations. It would allow us to simulate a wider variety of environments and tasks, leading to a more robust and adaptable robot operation.

In terms of alternate methods, we could have incorporated obstacle detection using sensors, such as cameras. This approach would involve creating an obstacle map in the Cartesian space, enabling the robot to dynamically adjust its path based on real-time data. It would have increased the complexity of our project but also the versatility and applicability of the robot.

In conclusion, this project has been a valuable learning experience in applying motion planning algorithms to solve a practical problem. Despite the challenges encountered, the successful completion of the task in both a real-world and simulated environment illustrates the potential of our approach. We look forward to building on this foundation, further exploring the exciting field of robotics, and pushing the boundaries of what can be achieved.

References

[1] S. Kagami, J. Kuffner, K. Nishiwaki, K. Okada, M. Inaba, and H. Inoue, "Humanoid arm motion planning using stereo vision and RRT search," presented at the Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), 2003, vol. 3, pp. 2167–2172.

[2] T. Tao, X. Zheng, H. He, J. Xu, and B. He, "An improved RRT algorithm for the motion planning of robot manipulator picking up scattered piston," presented at the 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC), 2018, pp. 234–239.

[3] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. D. Konidaris, "Robot Motion Planning on a Chip.," presented at the Robotics: Science and Systems, 2016, vol. 6.

[4] K. Elissa, "Title of paper if known," unpublished.S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," Int. J. Robot. Res., vol. 30, no. 7, pp. 846–894, 2011.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.K. Wei and B. Ren, "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm," Sensors, vol. 18, no. 2, p. 571, 2018.

[6] S. M. LaValle, Planning algorithms. Cambridge university press, 2006.

[7] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," Int. J. Robot. Res., vol. 21, no. 12, pp. 999–1030, 2002.

[8] A. Saxena, L. Wong, M. Quigley, and A. Y. Ng, "A vision-based system for grasping novel objects in cluttered environments," presented at the Robotics Research: The 13th International Symposium ISRR, 2011, pp. 337–348.

[9] Khatib, Oussama. "Real-time obstacle avoidance for manipulators and mobile robots." International journal of robotics research 5.1 (1986): 90-98.

[10] LaValle, Steven M. "Rapidly-exploring random trees: A new tool for path planning." Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998.

[11] Saha, Sourav, et al. "Collision avoidance in robotic manipulators using RRT-based motion planning and deep learning-based obstacle avoidance."

[12] Karaman, Sertac, and Emilio Frazzoli. "Sampling-based algorithms for optimal motion planning." The International Journal of Robotics Research 30.7 (2011): 846-894.

[13] Levenberg, David, and Omri Rand. "Manipulator path planning with RRT-connect." Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. 2006 IEEE International Conference on Robotics and Automation. IEEE, 2006

[14] Lin, Yongheng, et al. "A Real-Time Pick-and-Place Framework for Robot Manipulators with Kinematic Constraints." IEEE Robotics and Automation Letters 5.2 (2020): 2078-2085.

[15] Robotics and Autonomous Systems 134 (2020): 103642.

[16] 8.Corke, Peter. "Robotics, vision and control: fundamental algorithms in MATLAB." Springer, 2017.