Adaptable Real-Time Surface Painting Robot

ENPM662

Aakash Dammala, Akshun Sharma, Matt Kalavritinos, Pranav Koli

1. Introduction

1.1 - Motivations

In modern manufacturing, automation has become essential for improving efficiency, precision, and product quality while reducing costs and minimizing hazardous tasks for human workers. Robotic systems are widely used in industries such as automotive, aerospace, shipbuilding, and consumer-product manufacturing, where repeatability and accuracy are critical. Applications such as welding, surface polishing, assembly and painting all benefit from robots as they can operate with consistency, handle complex routines, and avoid health risks associated with chemical exposure or high stress manual labor. As manufacturing demands continue to grow, the need for reliable, high-performance automation is becoming even more important.

1.2 - Application

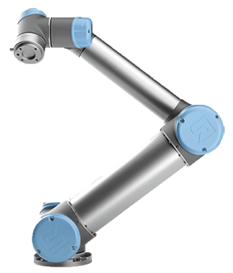
While industrial robots can paint predefined parts with excellent precision, they traditionally do not handle arbitrary objects. This limits their flexibility and prevents them from adapting to new or variable parts without reprogramming. The motivation behind our project is to overcome this limitation by enabling a robot to paint unknown objects. Our system scans the object, reconstructs its surface geometry, computes surface normals, and autonomously generates a painting trajectory tailored to that specific part. This adaptable, perception driven approach will operate in real-time and allow systems that are traditionally limited in their use cases to become more general-purpose painting robots.

2. Robot Details

2.1 - Robot Type

The robotic manipulator used in this project is a 6-degree-of-freedom (6-DoF) articulated robotic arm, which is a serial manipulator commonly used in industrial automation. Articulated arms consist of a series of rotary joints that provide flexibility and allow the tool head (end effector) to reach a wide range of positions and orientations within the robots workspace.

Justification: In this project we used the UR5 robot arm because of its good design and extensive usage in industrial automation. It has 6 DOF, which has enough DOF to reach any position and orientation, while not being redundant. And the manipulator size is big enough to reach the painting workspace requirement.



Universal Robotics UR5

The specifications for the UR5 are found below:

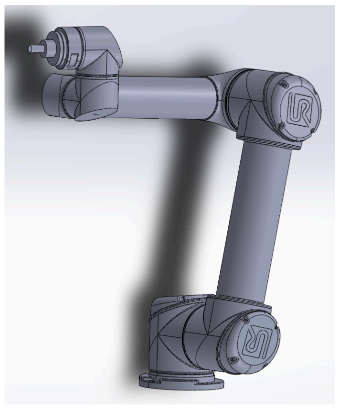
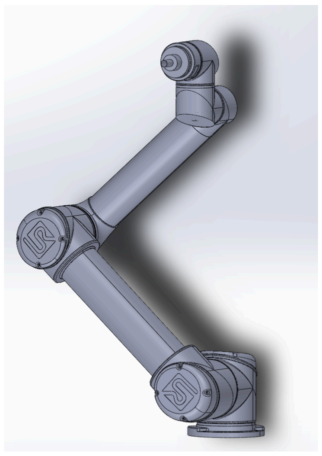
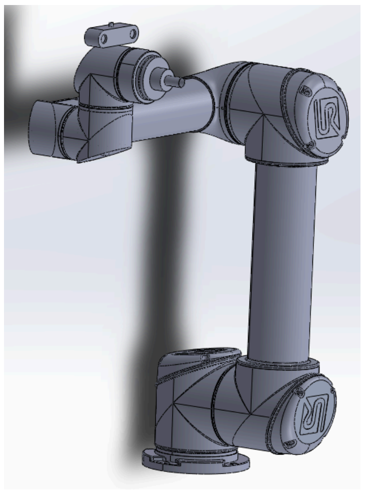
UR5 Manipulator	
DOF	6
Payload	5.0 kg
Reach	850 mm (33.5 in)
Repeatability	0.03 mm
Joint Range	All joints full 360°

2.3 - CAD Models

The CAD model of the UR5 robot arm was downloaded from Universal Robots [1]. Using this reference, a custom end effector was made to simulate a spray painting nozzle. An Intel RealSense camera was also attached to the wrist of the robot, however our project ended up not using the depth camera capabilities that it has to offer, and instead relied on a fixed 3D lidar system. The depth camera could be used in future iterations to further the capabilities of this project.

Using the SW2URDF plugin, we were able to export our CAD into URDF format. This package was then converted into XACRO format to be compatible with ROS2, and a dummy link connecting the base link and world was also added for ease of use in Gazebo.

We have added the CAD files in the CAD folder on our Github Repository.

 	
CAD without Intel RealSense	CAD with intel RealSense

2.4 - Frame assignment

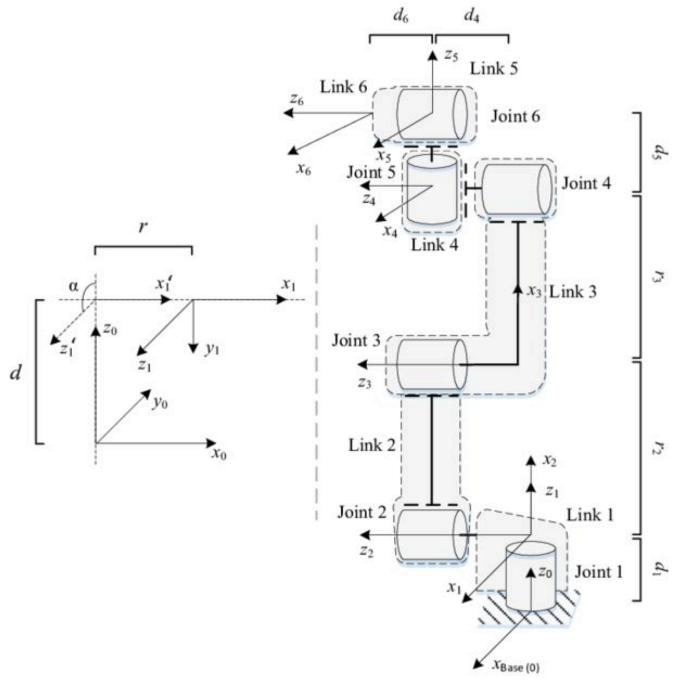


Fig. Frame Assignment for UR5 [9]

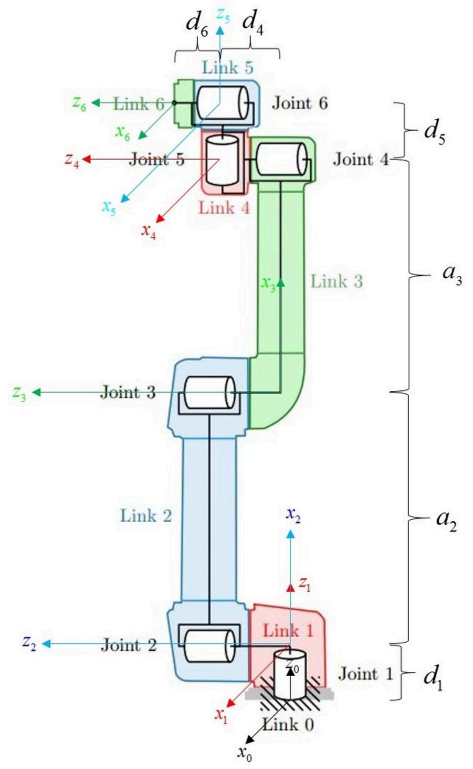


Fig. UR5 and Expected Joint Rotations [10]

3. Kinematics

3.1 - DH Parameters

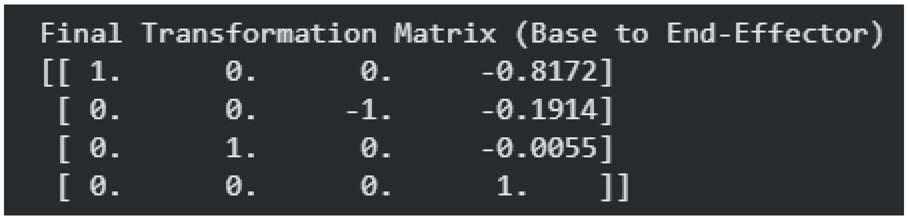
The following DH table was referenced from the UR5 documentation:

Axis	a [m]	α [rad]	d [m]	θ [rad]
1	0	$\pi/2$	0.89159	θ_1
2	-0.425	0	0	θ_2
3	-0.39225	0	0	θ_3
4	0	$\pi/2$	0.10915	θ_4
5	0	$-\pi/2$	0.09465	θ_5
6	0	0	0.0823	θ_6

3.2 - Forward Kinematics

Forward kinematics (FK) describes the mapping from the robot's joint angles to the position and orientation of its end effector. Based on the Denavit-Hartenberg parameters provided above, each link transformation can be represented using the standard homogenous transformation matrix.

Fig. : Final Transformation Matrix (Code Output)



```
Final Transformation Matrix (Base to End-Effector)
[[ 1.      0.      0.     -0.8172]
 [ 0.      0.     -1.     -0.1914]
 [ 0.      1.      0.     -0.0055]
 [ 0.      0.      0.      1.    ]]
```

$$
\begin{aligned}
A_i &= Rot_{z,\theta_i}\,\text{Trans}_{z,d_i}\,\text{Trans}_{x,a_i}\,Rot_{x,\alpha_i} \qquad (3.10)\\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&\quad \times \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Robot Modeling and Control - Spong [2]

Using this form, we can solve for $A_1 \rightarrow A_6$ by plugging in the respective values from the DH table, which will give us the transformation matrix for each link. Then by multiplying these together, we can find the overall transformation matrix from the base of the robot to the end effector.

$$T_0^6 = A_1A_2A_3A_4A_5A_6$$

This gives us a clean representation of where the end effector will be in space, in relation to the joint angles that we command.

3.3 - Inverse Kinematics

Inverse kinematics (IK) computes the joint configuration required to achieve a desired end-effector pose. Unlike forward kinematics which have a unique solution, the IK problem for 6-DoF serial manipulators has many valid solutions. To solve for the end-effector position corresponding to a target point in the world frame, we implemented a numerical IK solver based on the Jacobian pseudo-inverse (Moore-Penrose) method.

The Moore-Penrose method is implemented to solve for a joint space update that reduces the cartesian position error of the end effector. At each iteration, the forward kinematics are evaluated to obtain the current end effector position and the corresponding transformation matrices. These transforms are used to construct the full 6x6 Jacobian. More specifically,

$$J_{v_i} = z_{i-1} \times (o_n - o_{n-1})$$

Where J is the column of the Jacobian, z is the joint axis direction, and o_n , o_{i-1} are frame origins.

The linearized version of the kinematics problem is:

$$e = J\Delta q,$$

Where $e = p_{desired} - p(q)$ is the position error

Because J is not an invertible matrix, we apply the Moore-Penrose method. The numpy command linalg.pinv will compute the pseudo inverse for us, which is exactly what is done in our code. The joint update can then be computed as:

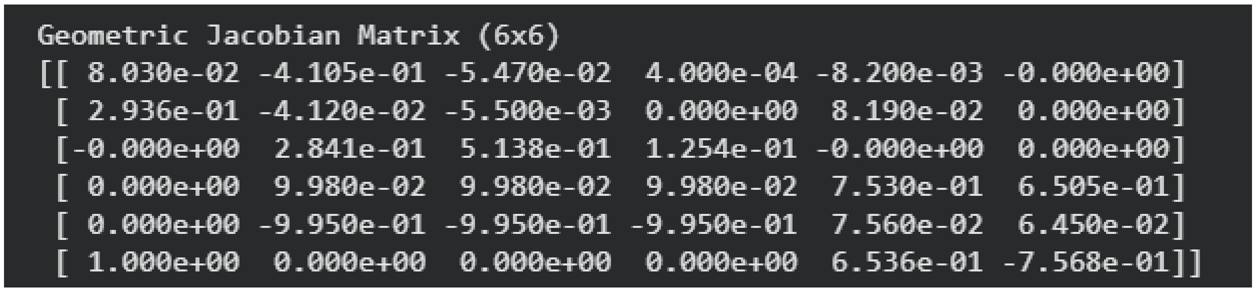
$$\Delta q = J^{\dagger} e$$

And we can apply this with a scalar gain to improve the stability:

$$q_{k+1} = q_k + 0.5\Delta q$$

This update is repeated iteratively within a Newton-Raphson framework until the norm of the position error falls below a specified tolerance or a maximum number of iterations is reached.

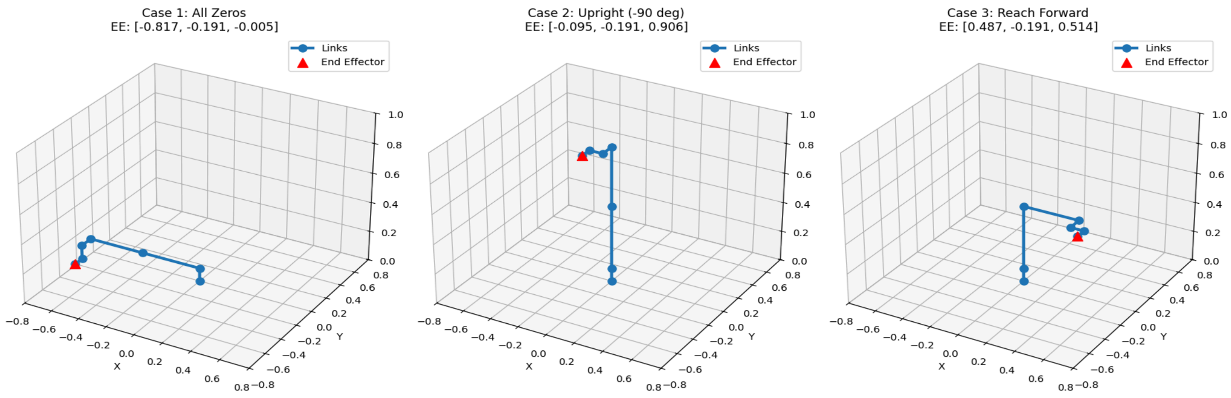
Fig. : Geometric Jacobian Matrix (Code Output)

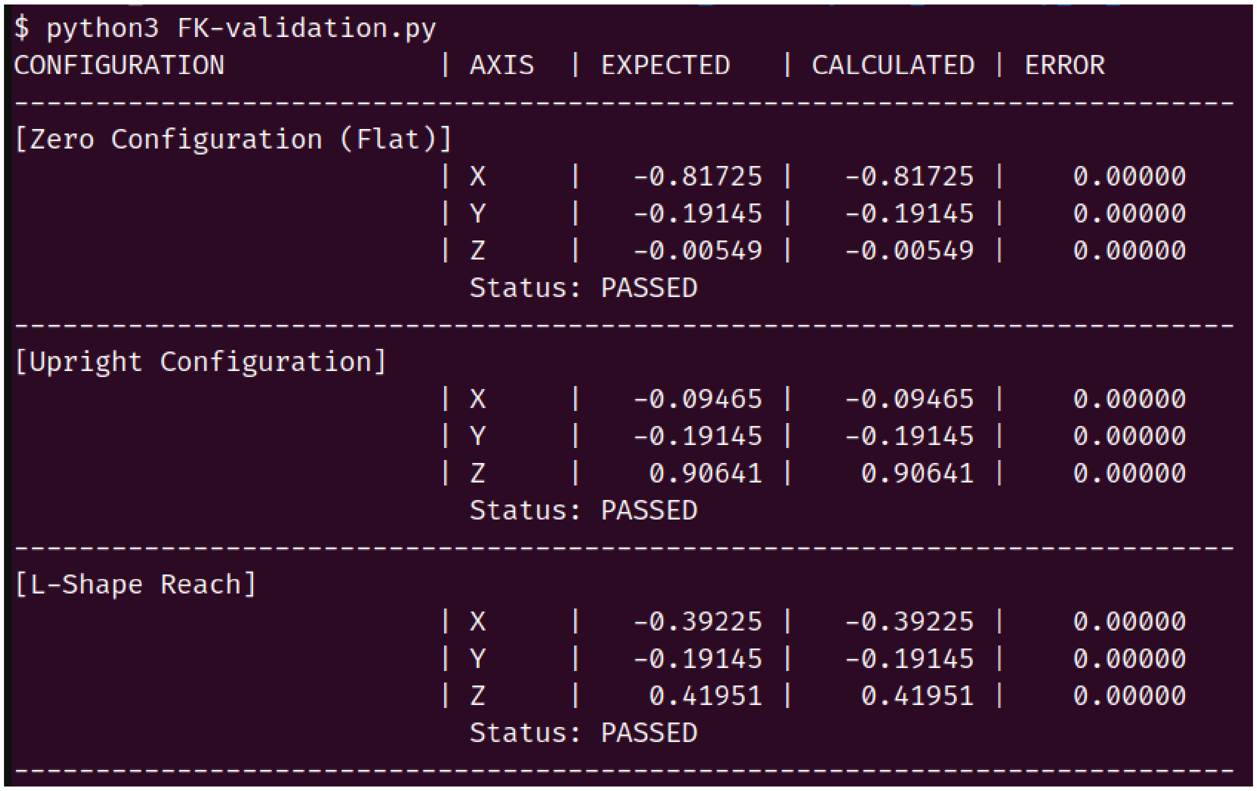


```
Geometric Jacobian Matrix (6x6)
[[ 8.030e-02 -4.105e-01 -5.470e-02  4.000e-04 -8.200e-03 -0.000e+00]
 [ 2.936e-01 -4.120e-02 -5.500e-03  0.000e+00  8.190e-02  0.000e+00]
 [-0.000e+00  2.841e-01  5.138e-01  1.254e-01 -0.000e+00  0.000e+00]
 [ 0.000e+00  9.980e-02  9.980e-02  9.980e-02  7.530e-01  6.505e-01]
 [ 0.000e+00 -9.950e-01 -9.950e-01 -9.950e-01  7.560e-02  6.450e-02]
 [ 1.000e+00  0.000e+00  0.000e+00  0.000e+00  6.536e-01 -7.568e-01]]
```

3.4 - Forward Kinematics Validation

Forward kinematics validation in 3 known positions, and verifying the end-effector positions with python code.
FK validation python code: ‘paint_cloud/my_ik_controller/my_ik_controller/FK-validation.py’
Github link: https://github.com/AakashDammala/paint_cloud





```
$ python3 FK-validation.py
CONFIGURATION              | AXIS   | EXPECTED    | CALCULATED | ERROR
---------------------------------------------------------------------------
[Zero Configuration (Flat)]
                           | X      |   -0.81725 |   -0.81725 |    0.00000
                           | Y      |   -0.19145 |   -0.19145 |    0.00000
                           | Z      |   -0.00549 |   -0.00549 |    0.00000
                             Status: PASSED
---------------------------------------------------------------------------
[Upright Configuration]
                           | X      |   -0.09465 |   -0.09465 |    0.00000
                           | Y      |   -0.19145 |   -0.19145 |    0.00000
                           | Z      |    0.90641 |    0.90641 |    0.00000
                             Status: PASSED
---------------------------------------------------------------------------
[L-Shape Reach]
                           | X      |   -0.39225 |   -0.39225 |    0.00000
                           | Y      |   -0.19145 |   -0.19145 |    0.00000
                           | Z      |    0.41951 |    0.41951 |    0.00000
                             Status: PASSED
---------------------------------------------------------------------------
```

3.5 - Inverse Kinematics Validation

Inverse kinematics validation using pose plots

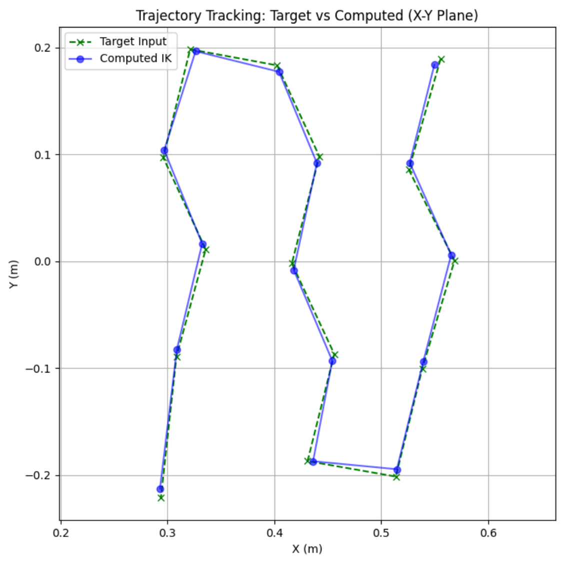


Fig. : End effector trajectory (in X-Y plane)

Green dashed line - input trajectory from the trajectory planner

Blue solid line - output trajectory from custom IK solver, to robotic manipulator

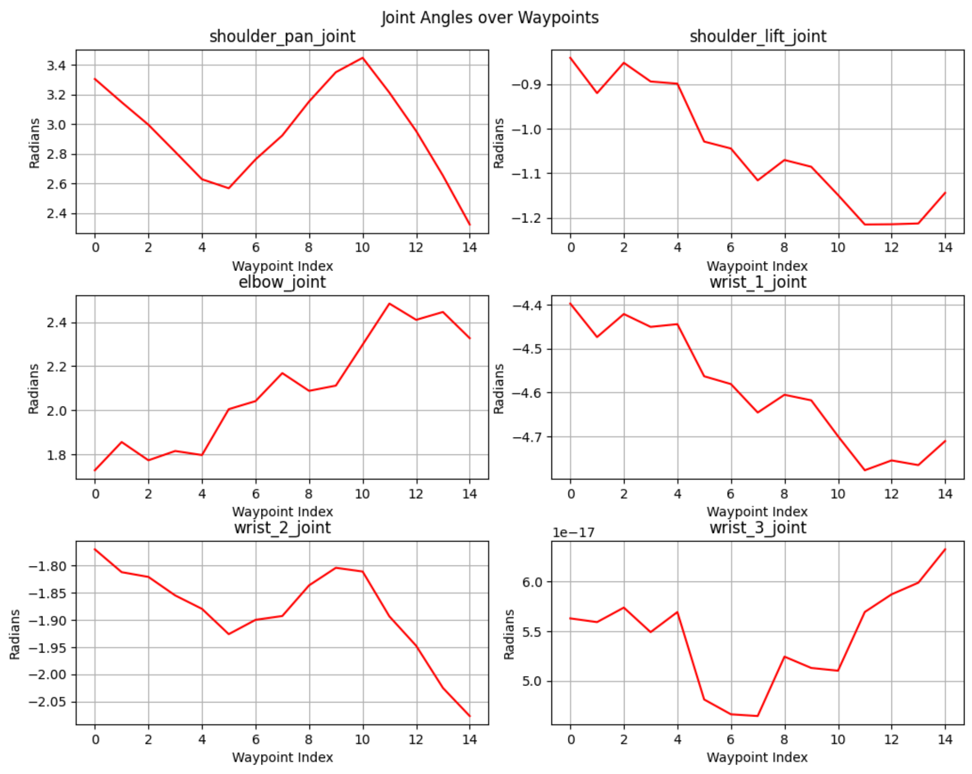


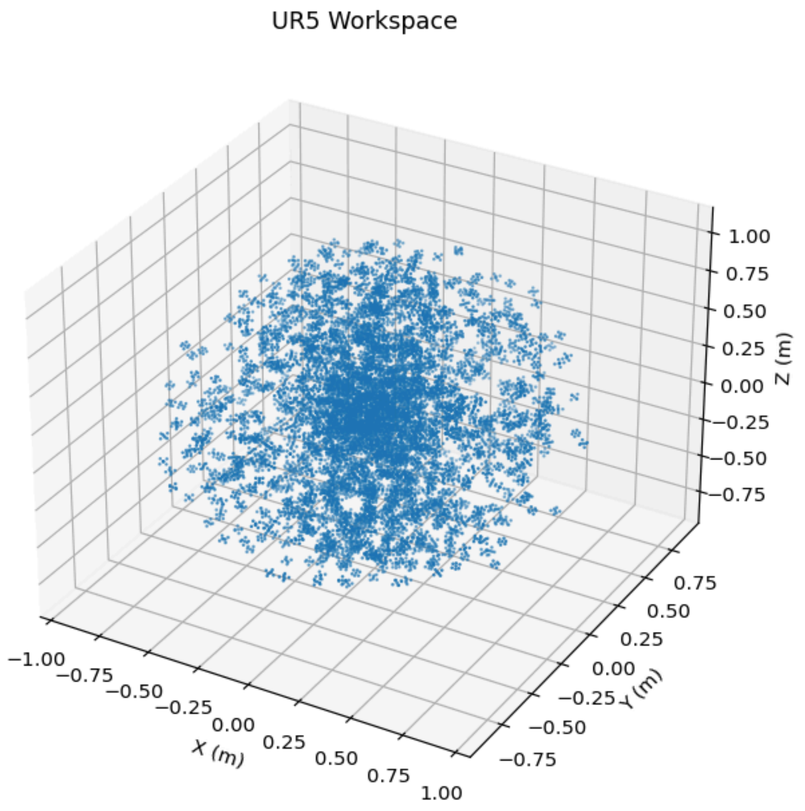
Fig. 6x Joint positions trajectory plot over following trajectory

Inverse kinematics validation using Gazebo - The end effector is following the planned trajectory closely, this can be verified visually.

4. Workspace Analysis

4.1 - Workspace Study

Below the reachable position workspace is plotted:



The workspace represents the reachable cartesian positions of the UR5 end effector without enforcing orientation constraints or collision avoidance, assuming the full joint range is available. Visualizing this region allows us to consider what size and type of objects the UR5 may be suitable for painting. We are able to quickly check with minimal effort if the surface will be able to be painted.

4.2 - Assumptions Made

Several assumptions were made throughout the development of this project First, the 3D LiDAR is assumed to be mounted in a known and fixed pose relative to the robot base frame. This allows transformation matrices to be applied sequentially from the LiDAR frame to the world frame, and subsequently to the robot base and end effector frames.

Next, no explicit workspace feasibility check is performed for the target painting surface. As a result, it is assumed that the region of interest lies within the reachable workspace of the robot. In practice, this assumption was enforced by testing only on small objects whose placement could be visually verified to be within the robots reachable region.

Additionally, there was no use of MoveIT, FCL or any geometry based collision detection. All motions are assumed to be valid and not to interfere with the environment in an adverse way.

Finally, it is assumed that all joint positions are exact. That is to say, there is no internal PID or other feedback controller that will command the joint angles to converge on a position, even if there is some error.

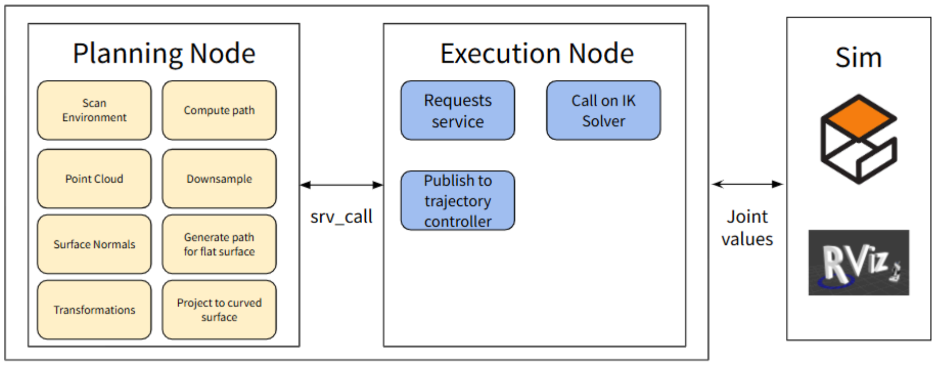
4.3 - Controllers used

Inverse kinematics controller has been used, each waypoint calculated is passed on to this IK solver / controller, which in turn generates the joint values. These joint values are passed on to gazebo simulation using urdf's Joint trajectory controller.

5. Controls and Simulation

5.1 System Architecture

The overall system architecture for our project consisted of two main nodes, the planning and the execution, which each interact with the simulations and visualizations done in Gazebo and Rviz respectively. The planning node handles the 3D LiDAR scan which produces a point cloud that is used to extract the surface normals of the surface. A 2D painting trajectory is then projected onto the 3D surface and a path that follows this trajectory is produced. The execution node can then call on a service from the planning node that will feed in the desired end effector positions. These values are fed into the IK solver and published to the position controller. These each interact with Rviz & Gazebo, where we can visualize and simulate the results.



5.2 Scanning

5.2.1 Scanning the environment

The first step of the process is to scan the part on the table. This step involves scanning the part using a Lidar scanner fixed above the bench, and acquiring the 3d pointcloud from it.

5.2.2 Processing Point Clouds

The raw point cloud is treated to make it both geometrically useful and computationally manageable.
Downsampling: To increase the computational efficiency of the path planning algorithms, the subscribed point cloud is instantly downsampled to lower the number of data points.
Coordinate Transformation: Next, the down-sampled point cloud is converted from the local sensor frame (the frame of the LIDAR) to the world frame. Accurate path planning depends on this uniformity.

5.2.3 Calculating Surface Normals

The surface orientation needs to be calculated in order to allow planning that complies with the object's geometry:

Principal Component Analysis (PCA): To extract the surface normals, the system applies Principal Component Analysis to the down-sampled point cloud.

Surface Normal Extraction: The vector for the surface normal at a given location on the point cloud is obtained from the third eigenvector that arises from the PCA algorithm for that local area. The 2D travel plan must be adjusted to the 3D curved surface using these normals.

The scanning module's final output, which is prepared to be sent to the Path Planning module, is the cleaned, converted, and normal-annotated point cloud.

5.3 Path Planning

The path planning module calculates the best course for the robot's end-effector using the processed point cloud and surface normal data from the scanning phase. This module is in charge of creating a thorough, safe path that covers the necessary surface area while preserving the proper tool orientation.

5.3.1 2D Projection and Planning

To make the first trajectory computation easier, the path generating process starts in two dimensions.
2D Pose Generation: To cover the desired surface area within a 2D polygon, the system first creates a series of poses (position and orientation). A zigzag motion is frequently the initial path pattern.
Adaptation to 3D: The generated 2D plan is modified to reflect the actual 3D geometry by projecting the 2D points onto the 3D point cloud of the scanned object.

Offsetting: The final poses along the calculated surface normals are subjected to a specific value, paint_distance. This offset is crucial for tasks like painting or inspection where the tool needs to be kept at a precise distance from the object's surface.

Coordinate extraction: A set of Cartesian coordinates is stored for each point's updated 3D trajectory.

5.4 Gazebo and Rviz

Youtube video link: https://www.youtube.com/watch?v=ujLB43V4m5M

In order to create, test, and debug the path planning and execution system without requiring real hardware, the project makes use of industry-standard robotic simulation and visualization tools called Gazebo and Rviz.

The robot and the workpiece communicate in a controlled virtual environment using Gazebo, a physics-based simulator. Its purpose is to simulate the system's physical parts, such as the robotic manipulator and the sensor that collects data.

LIDAR Integration: To accurately replicate the scanning procedure, a 3D LIDAR sensor is integrated into the Gazebo environment. The raw point cloud data that the planning nodes use is the result of this simulated sensor.
A 3D visualizer called Rviz (ROS Visualization) is used to track the system's current state, ensure data integrity, and visually evaluate the accuracy of the planned trajectory prior to physical execution.

Point Cloud Visualization: To ensure that the environment is appropriately represented digitally, Rviz is used to show the point cloud (pcd) data obtained from the scanning process.

Trajectory Visualization: The anticipated Cartesian coordinates and orientations that the end-effector will follow can be examined in real time by seeing the generated path as a pose array.

6. Project Challenges

6.1 - Problems Encountered

Implementing the robot model using the **URDF** (Unified Robot Description Format) and **XACRO** formats proved to be difficult. Correctly defining the robot's kinematics, visuals, and collision properties within these files required extensive debugging.

Implementing the **IK solver** was challenging. The numerical Jacobian method required careful tuning and handling of singularities or joint limits to ensure a valid and smooth trajectory of joint values.

The major challenge was integrating the planning node which did the sensing and path planning and the inverse kinematics node. Both of these nodes were developed independently by different teammates and it was hard to get the two to communicate.

7. Conclusions and Future Work

7.1 - Conclusions

After a lot of work in figuring out the integration, we were able to move the robot along a curved surface and were able to move the end effector along a zig zag painting trajectory which was the objective of our project. We added the controllers on the UR5 robotic arm and implemented Inverse Kinematics to test run it on Gazebo. Parallely we were testing the scanning strategy and making sure to project the surface normals on the curved surface.

7.2 - Future Work

There are several directions in which this project could be extended and improved. From a control and motion planning standpoint, integrating our system with MoveIt would provide more reliable inverse kinematics (IK) solutions and enable native collision avoidance. One limitation of our current approach is that the IK solver does not consistently select the correct elbow configuration. While the solver often returns a feasible end-effector position, it may do so by commanding a configuration that causes the robot to collide with the ground. We also observed cases where the solver produced joint angles that exceeded the limits specified in the URDF. Incorporating MoveIt's kinematics plugins and constraint handling would address these issues, enforce joint bounds, and provide more predictable and repeatable results.

Another important limitation of the current controller is that we implemented position control only, without explicit orientation control over the end effector. For surface finishing or painting, end-effector orientation is essential as the tool must point toward the surface at the desired angles at all times. Future work should include either adapting the IK solver to handle orientations and elbow configurations, or using MoveIt. Additionally, we have only implemented open loop control, we command the joint angles and the joints move accordingly. Closed loop control of the end effector via PID or other more advanced controllers could be implemented to minimize the error of the end effector location.

On the trajectory generation side, we made two simplifying assumptions to facilitate initial integration and testing. First, we assumed a fixed, global zig-zag pattern for surface coverage. Second, we generated this pattern inside a large polygon that extended well past the object. A more robust approach would directly compute the 2D projection of the object's surface and generate coverage paths constrained to this region. This would reduce unnecessary motion, prevent painting outside the target area and minimize both time and material waste.

Finally, the last major limitation of our approach is the fixed base, which significantly restricts the reachable workspace. Adding a linear rail or gantry would provide additional degrees of freedom and allow the system to paint larger or more complex surfaces. Overall, these enhancements would lead to a more capable, efficient, and reliable autonomous painting solution

8. References

- [1] P. M. Kebria, S. Al-wais, H. Abdi, and S. Nahavandi, "Kinematic and dynamic modelling of UR5 manipulator," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Budapest, Hungary, 2016, pp. 004229–004234. doi: 10.1109/SMC.2016.7844896.
- [2] "DH Parameters for calculations of kinematics and dynamics," Universal Robots. [Online]. Available: https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/
- [3] "Stereo Depth Camera D435," RealSenseAI. [Online]. Available: https://realsenseai.com/stereo-depth-cameras/stereo-depth-camera-d435/
- [4] W. Lin, A. Anwar, Z. Li, M. Tong, J. Qiu, and H. Gao, "Recognition and pose estimation of auto parts for an autonomous spray painting robot," *IEEE Trans. Ind. Inf.*, vol. 15, no. 3, pp. 1709–1719, Mar. 2019. doi: 10.1109/TII.2018.2882446.
- [5] S. Nieto Bastida and C.-Y. Lin, "Autonomous trajectory planning for spray painting on complex surfaces based on a point cloud model," *Sensors*, vol. 23, no. 24, p. 9634, 2023. doi: 10.3390/s23249634.
- [6] J. M. Casanova, "Simulation and planning of a 3D spray painting robotic system," Ph.D. dissertation, Univ. of Porto, Portugal, 2021.
- [7] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, 2nd ed. Hoboken, NJ, USA: Wiley, 2020.
- [8] "Robot UR5 STEP file (CB Series)," Universal Robots. [Online]. Available: https://www.universal-robots.com/download/mechanical-cb-series/ur5/robot-ur5-step-file-cb-series/

- [9] R. Galin, “Mathematical Modelling and Simulation of Human-Robot Collaboration,” in *2020 International Russian Automation Conference (RusAutoCon)*, Sochi, Russia, Sep. 2020, pp. 112–116, doi: 10.1109/RusAutoCon49822.2020.9208040.
- [10] P. M. Kebria, S. Al-wais, H. Abdi, and S. Nahavandi, “Kinematic and dynamic modelling of UR5 manipulator,” in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Budapest, Hungary, Oct. 2016, pp. 4229–4234, doi: 10.1109/SMC.2016.7844896.

9. Github Link

https://github.com/AakashDammala/paint_cloud