Vision Guided Robotic Manipulation for Satellite Repair and Maintenance

Ian Pylkkanen, Sudarshan Thirumalai, Lachlan Moore

Rensselaer Polytechnic Institute

MANE 4560/6120

2020
December

Executive Summary

This report details the design and simulation of a 7 degree of freedom robot arm for use in satellite operations. Derivations and descriptions of modeling, kinematics, vision guidance, and control are thoroughly discussed in the sections to follow. Through the use of MATLAB programming, these components are all successfully integrated to produce a uniform end product. This project is successful in laying the framework for more in depth research into the feasibility and implementation of automated satellite maintenance and repair.

Contents

List of Figures

List of Tables

1 Introduction

The research team at Team 2 Robotics was tasked with designing, modeling, and simulating a satellite repair robot. The team's primary academic and professional background is in the field of aerospace engineering. Thus, pursuing a robotics application in the aerospace industry was desirable. The team centered upon the idea of implementing a robot that could complete simple satellite repair tasks with the possibility of assisting astronauts in more complex undertakings. The robot has desired specifications of a vision guidance system and a multi degree of freedom arm to effectively enable a multitude of satellite repair operations.

1.1 Motivation

With the prevalence of satellites in orbit today, it is a matter of when and not if something will go wrong. Whether it is due to a collision with space debris or a system failure, it is almost a guarantee that at some point during the life cycle of a satellite, it will need on-orbit repairs. The manual on-orbit repair and assembly of satellites by astronauts is an extremely rigorous and dangerous task. As a result, the use of vision guided robots to accomplish this is extremely important. For years the National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA) have used robots to perform rudimentary tasks to help repair and construct satellites and space stations in orbit. The use of robots greatly reduces the risk to astronaut lives while performing such tasks, also increasing efficiency. Robots do not need to return to a spacecraft after a few hours, they can continuously operate as necessary.
The motivation for this project was to develop a system that can be used in place of astronauts for on-orbit satellite construction, maintenance, and servicing. This will provide a cheaper and safer alternative as well as work to clean up Earth's orbit in the near future.

1.2 Goals

The following goals were set prior to beginning the project to gauge the performance and success of the implementation of the robot. Throughout the process, the end-effector capabilities were discarded in favor of a more robust vision guidance and trajectory control system. For this reason, the "Reach" goal was not met, though the "Baseline" and "Target" goals were. The results meeting these goals will be discussed further in this report.

- Baseline
  The baseline goal will be to program a robot arm to locate a screw, move to the appropriate location, torque the screw and repeat.

- Target
  The target is to design and implement a vision guided robot to attach a

flat panel using screws in the plane defined by the x, and y-axes. Screw holes will be in a predetermined array, and vision will be used to locate and double check screw holes.

- Reach
  The vision system will be able to detect arbitrary screw hole locations on the flat panel, and determine the number of screws necessary. The end-effector will also have variable torque capabilities.

2 Approach

A high level overview of the robot functions are given below. The chart is chronological, as each consecutive section depends on the one previous to it.

1. Imaging

   The robot is fitted with two pinhole cameras, each of which captures images of the screws or other desired waypoints. It then calculates the position with 2 images and the predefined camera parameters.

2. Trajectory

   The robot end effector is manually manipulated to generate a rough path. This step, while also determining viable waypoints, avoids collisions with the various obstacles surrounding the robot. The robot then generates a trajectory to smooth the motion between the pre-selected waypoints.

3. Simulation

   This stage is largely circumstantial due to the inability to build a physical robot. The simulation runs through the tasks that the physical robot would perform based on the given trajectory plan.

3 Robot Modeling

The design of the robot is dissimilar to any pre-existing robot geometry that has been discussed in class. The preliminary robot geometry was taken from previous research done on a servicing system designed for geostationary orbit [1]. The robot is a 7 degree of freedom (DOF) arm with 7 serial revolute joints. The joint configuration and DH parameters are discussed in the following sections.

3.1 POE Parameters

The product of exponentials (POE) method is a convention for the definition of robot geometry. It gives the dimensions of a serial n-link arm in the base frame while the robot is at the 0 joint angle configuration. The rotational axis

for the joints is defined by H = $[h_1...h_i]$ while the dimensions of the links are defined by P = $[P_{01}...P_{nT}]$. The origin of each joint O_i lies on the axis h_i. Figure 1 shows the model POE definitions. The parameters in the ϵ_0 frame are defined by equations (1) and (2).

$$H = \begin{bmatrix} ez & ey & ez & ez & ex & ez & ex \end{bmatrix}_{\epsilon_0} \tag{1}$$

$$P = \begin{bmatrix} 400 * ez & 320 * ey - 720 * ex & -720 * ey & zv & 720 * ex & -320 * ez & -475 * ex & zv \end{bmatrix}_{\epsilon_0} \tag{2}$$

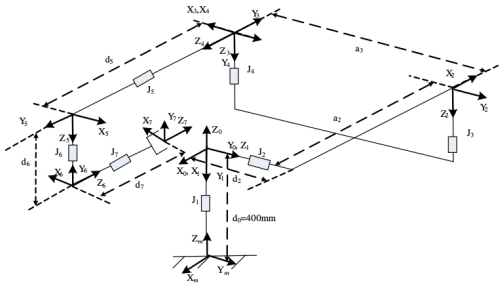

Figure 1: Robot Geometry: Product of Exponentials

3.2 Denavit-Hartenberg Parameters

Denavit-Hartenberg (DH) parameters are often used to make modeling and implementation into prgrams such as MATLAB easier. DH parameters are commonly used in industrial robotics as they are an easy and effective way to find homogeneous transformation matrices [2]. They are defined from the POE coordinate systems and represented by four parameters:

θ_i: Angle between x_{i-1} and x_i axes about the z_{i-1} axis.
α_i: Angle between the z_{i-1} and z_i axes about the x_i axis.
a_i: Distance between coordinate frame $i-1$ and i along the x_i axis.
d_i: Distance between coordinate frame $i-1$ and i along the z_i axis.

The DH parameters of the 7 degree of freedom robot arm for this project are defined in Table 1.

Link i	θ_i	α_i	a_i (mm)	d_i (mm)
1	0°	−90°	0	0
2	180°	90°	720	320
3	−90°	0°	720	0
4	0°	90°	0	0
5	180°	90°	0	720
6	180°	−90°	0	320
7	0°	0°	0	475

Table 1: DH Parameters of 7-DOF Robot Arm

With the DH parameters defined, it was necessary to move to MATLAB's Rigid Body Tree to begin modeling the robot.

3.3 MATLAB Rigid Body Tree

The geometry and parameters previously defined were translated into MATLAB by using a rigid body tree model from the Robotics Toolbox. The rigid body tree, as the name suggests, generates a model from rigid links and joints. Each joint rotation is configured to its respective link, and the rigid body tree keeps track of the relationship between all the links and joints. The model of the robot arm in MATLAB's Rigid Body Tree is shown in Figure 2.

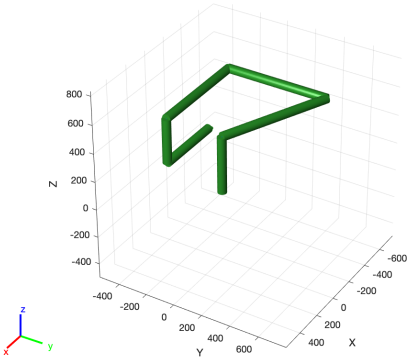

Figure 2: Robot Arm in Rigid Body Tree

Each link of the robot is modeled as a cylindrical collision body. This will allow the robot to avoid colliding with itself later on when collision avoidance and path planning are implemented.

3.4 Collision Bodies

Along with the collision bodies of the robot arm, three more collision bodies were defined in the space. Two cylindrical pillars were added to simulate support beams, and a rectangular box was added as the base structure for the robot arm. The bodies were added in order to simulate structures that the robot arm could run into on any given satellite. To do so, the beams and base were created using the provided MATLAB functions *collisionCylinder* and *collisionBox*. These functions create the structures as collision bodies. They are shown along side the robot in Figure 3.

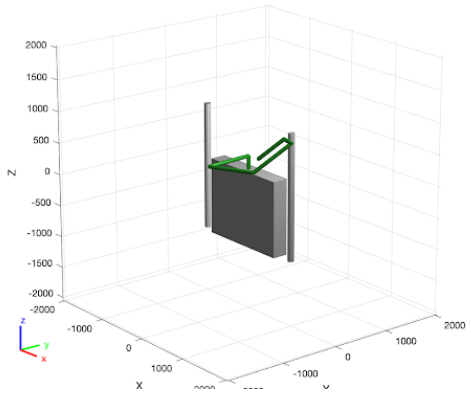

Figure 3: Collision Bodies

The robot, however, will not avoid the collision bodies simply because they are defined as such. Instead a collision avoidance study needs to be performed. This study is detailed in Section 6.1.

4 Kinematics

4.1 Forward Kinematics

Forward kinematics is the determination of end-effector position and orientation based on an input of joint angles. This solution can be determined from the product of exponential parameters laid out in Section 3.1. The algorithm was taken from *Lecture 8 Notes* and is as follows [3].

Algorithm 1: Forward Kinematics

Result: (P_{OT}, R_{OT})
Define the base frame ε_0 and origin O_0;
Define the zero position, the configuration with each joint angle set equal to 0, and all frames aligned;
Choose origins of the ith body, O_i, along the axis h_i;
while *chain is in the 0 position* **do**
    Find h_i;
    Find $p_{i-1,i}$;
end
Apply forward kinematics equations, (3) (4)

$$R_{0T} = R_{01}(q_1)R_{12}(q_2).....R_{n-1,n}(q_n)R_{nT} \tag{3}$$

$$P_{0T} = P_{01} + R_{01}(q_1)P_{12}..... + R_{n-1,n}(q_n)P_{nT} \tag{4}$$

This solution was implemented into MATLAB code by Dr. John Wen and was used for this project [4].

4.2 Inverse Kinematics

Inverse kinematics is the process of determining a joint angle solution from a given end effector position and orientation. Though this is simply the reverse of forward kinematics, the solution is not nearly as trivial. There are many algorithms that offer solutions for the inverse kinematics problems. For an n-link arm there is more than one unique solution for viable joint angles. In the case of a 6 DOF arm for example there are 8 unique solutions. In this scenario, an analytical solution can easily be obtained from the canonical sub-problems [5]. For the application of a 7 DOF arm there are redundant solutions, leading to difficulty and high computational cost in finding an analytic solution. For this reason, 7 degree inverse kinematics solutions are often iterative in their approach.

There are a number of existing algorithms that can find solutions for this problem. Some common algorithms include BFGS unconstrained optimization, Levenberg-Marquardt (LM) inverse least squares, or even particle swarm optimization [6, 7]. For more information on the BFGS algorithm for optimization, the reader can see [8, 9]. This project chose to use the LM inverse least squares method due to the convenient package already implemented into the MATLAB robotics toolbox [7].

The LM algorithm is a method of gradient based optimization for converging to a solution. This is an error damped least squares algorithm meaning that the solution will not escape from a minimum. The downside to the LM algorithm is the extended time for convergence, though the solution will converge faster if the initial guess is close to the minimum. Moving a robot along a trajectory path often provides good initial guesses for the next waypoint. This is an area to explore in future research. The major benefit, and reason for using this solution, is the ability to apply the algorithm to varying robot geometry. For example, the subproblems must be re-solved for new geometry, while the same LM algorithm can be applied. The full derivation for this algorithm is lengthy and the method is presented here with the understanding that the reader can seek other studies on the calculations. For more information of the full derivation and implementation, see [7].

5 Vision Guidance

The purpose of the vision guidance system in this application is to determine the target locations of potential satellite defects and calculate the inertial coordinates in the robot frame. Using a set of stereo cameras allows for a position to be determined from a single image from each camera. This information is then fed into the path planning and trajectory generation to calculate the robot motion.

The vision guidance system consists of two pinhole cameras fixed to the body of the robot. Section 5.1 discusses the solution of target positions from cameras that were simulated in this project.

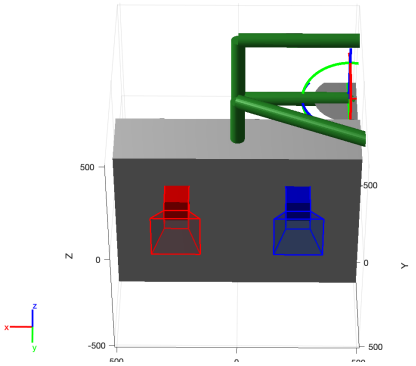

Figure 4: Stationary Cameras

5.1 Stereo Pinhole Camera Solution

In order to simulate the stereo camera system that is implemented here, no real cameras were used. The two cameras were simulated using MATLAB with predefined parameters. Using cameras fixed to the robot base gave known extrinsic parameters which consist of the position and orientation in the robot inertial frame. These are defined by Equation 5. Both cameras were set to have the same initial intrinsic parameters as defined in Table 2, assuming perfect calibration.

$$T = [R_{co} P_{co}] = \begin{bmatrix} 1 & 0 & 0 & \pm 0.25 \\ 0 & 1 & 0 & 0.175 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

Camera	u_0	v_0	f	f/ρ_h	f/ρ_w	W	H
1	640	512	.0075	750	750	1280	1024
2	640	512	.0075	750	750	1280	1024

Table 2: Camera Intrinsic Parameters

After these parameters are known, a target position can be determined from a single image from each camera. This problem is visualized in Figure 5.

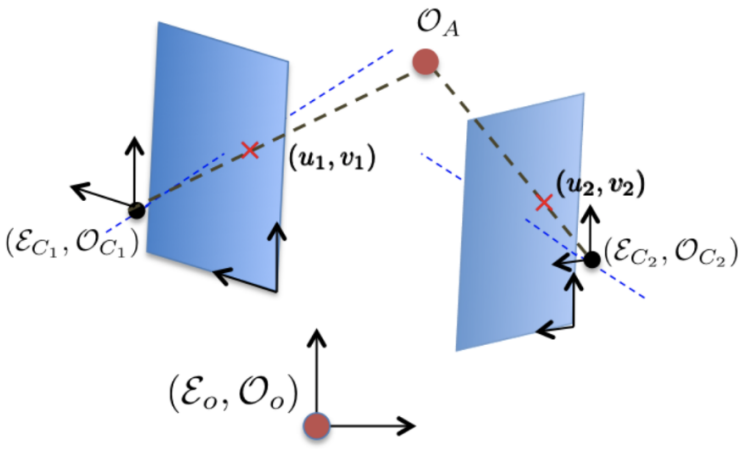

Figure 5: Stereo Pinhole Camera Problem

The goal of this problem is to solve for the vector $\vec{P}_{OA}$, which is the location of the target relative to the robot in the robot inertial frame. This is equivalent to the sum of vectors $\vec{P}_{OC}$ and $\vec{P}_{CA}$, or the distance from the world frame to the camera and the camera to the target. $\vec{P}_{OC}$ is defined from the already determined extrinsic camera parameters, so the necessary component is $\vec{P}_{CA}$. In order to solve for a target position from images from pinhole cameras, the image coordinates of the target must be known. The following solution assumes this is the case, and the image processing will be discussed in Section 5.3. The pinhole camera model defines the following relationship.

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} f/_w & 0 & u_0 \\ 0 & f/rho_h & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{CA} \\ y_{CA} \\ z_{CA} \end{bmatrix} \tag{6}$$

For a pinhole camera model with 2 unique cameras, the solution for target position in 2 images take the form of Equation 7.

$$\begin{bmatrix} u_1 a_{w_1}^T - a_{u_1}^T \\ v_1 a_{w_1}^T - a_{v_1}^T \\ u_2 a_{w_2}^T - a_{u_2}^T \\ v_2 a_{w_2}^T - a_{v_2}^T \end{bmatrix} \vec{P}_{oA} = \begin{bmatrix} b_{u_1} - b_{w_1} \\ b_{v_1} - b_{w_1} \\ b_{u_2} - b_{w_2} \\ b_{u_2} - b_{w_2} \end{bmatrix} \tag{7}$$

$$a_{u_i}^T = e_x^T K_i R_{c_i o} \vec{P}_{OA} \tag{8}$$

$$b_{u_i} = e_x^T K_i \vec{P}_{c_i o} \tag{9}$$

Similarly, equations 8 and 9 can be used to solve for the remaining a and b terms. This resolves all unknown terms in order to determine camera target position from two images by stereo cameras. This solution can be implemented to accept real image coordinates post MATLAB processing. Section 5.3 will discuss this process.

5.2 Camera Calibration

To more accurately simulate real cameras, calibration was performed on the two simulated cameras with an introduced pixel error. The cameras were calibrated using supplied MATLAB function *camcalib*. This creates a set of target points on test images and ultimately estimates the intrinsic camera matrix K. When comparing the estimated matrix to the perfectly simulated cameras, the error was less than 2% which was deemed acceptable for use in target finding and trajectory generation. For further further reading on the derivation of the camera calibration code, the reader can see [10].

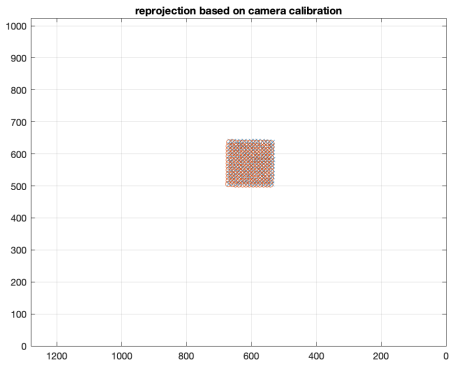

Figure 6: Camera Calibration Target Reprojection

5.3 MATLAB Image Processing

To process the images as real cameras would, MATLAB's *Camera Calibration Toolbox* was used. Real images were used for a situation that could be applicable to a satellite repair robot. In this case, images of screw heads were used both for their relevance and the simplicty of shape geometry detection. The process was broken into 3 major steps: (1) Simulate a set of images taken from two stereo cameras. (2) Process the images looking for circles or screw heads. (3) Determine the image coordinates of the points of interest to pass into the algorithm discussed in Section 5.1.
The first step in this process was to simulate a set of images that would have been captured from the stereo cameras. Figure 8 shows the set of images that were used for the simulation.

(a) Camera 1 (b) Camera 2

Figure 7: Stereo Images from Cameras

Using the MATLAB built in function *imfindcircles*, circles in an image are detected using a Hough transform. For more information on this method, see MATLAB documentation for the *Image Processing Toolbox* [11]. The result of the Hough transform gives the center and radius of each detected circle. The only value of concern is the center, as these are the image coordinates that will be passed into the above method.

(a) Camera 1 (b) Camera 2

Figure 8: Recognized Screw Heads in Images

After using the discussed algorithm, the projected points are shown in the 3D robot space in Figure 9. For the path planning and trajectory in further sections, simulated images with 3 points of interest were used.

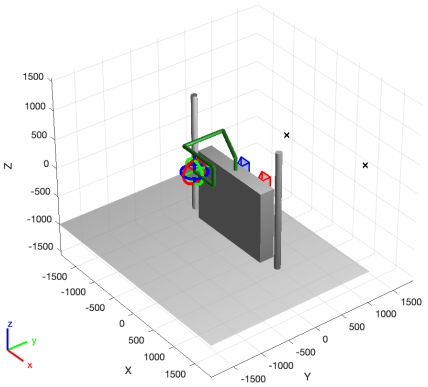

Figure 9: Project Points From Camera Images

6 Path Planning and Trajectory Generation

6.1 Collision Avoidance

The collision bodies implemented in Section 6.1 were added to give the robot "obstacles" to avoid. However, as previously mentioned, the robot will not avoid the bodies simply because they are collision bodies. Instead a collision avoidance study must be performed. The most effective way to do this is to perform a free space calculation. The goal of a free space calculation is to determine the acceptable values of joint angles that the robot can move to without colliding with itself or any other collision bodies. To perform this analysis, a free space plot is required. The plot is illustrated such that the invalid joint angle configurations are marked with red points. In this type of plot, the area with no points (or the free space) represents the locations of valid joint configurations. The free space plot for the robot arm is displayed in Figure 10.

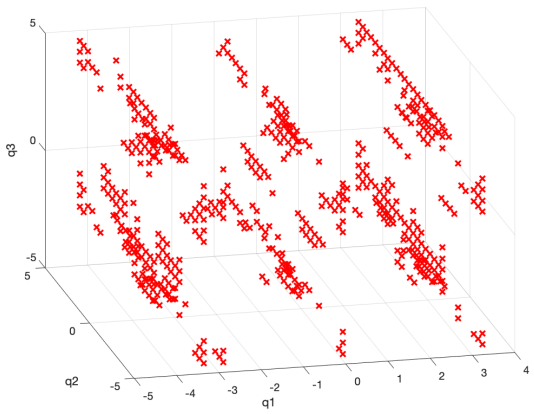

Figure 10: Free Space for First 3 DOF

The free space graph is designed such that each axis of the graph is one joint angle. Because of this, it is impossible to plot a free space graph for more than three joint angles. As a result, the motion of the 7 degree of freedom robot arm cannot fully be determined by a free space plot. However, it can still be beneficial to understanding the general motion that the robot is allowed. To perform the free space calculation for the 7 degree of freedom robot arm, the last four joint angles are considered to be frozen, while the first three are given full range of motion. This allows for a 3 degree of freedom free space calculation. The calculation is carried out through three nested for loops in MATLAB, where each joint angle is given a range of values. At each configuration of joint angles, the provided MATLAB function *collisionCheck* is run to determine if the robot will collide with a collision body. If the configuration fails the check and the robot collides with one of the obstacles, a red mark will be plotted on the free space graph. While this free space calculation does not provide all the information necessary to determine the allowable motion of the robot, it does provide a good reference to begin such a determination.

6.2 Path Planning

The process of path planning is simply the process of choosing points, either in terms of joint angles or Cartesian coordinates, for the robot arm to visit. These points, known as waypoints, have to be chosen in such a way that the robot arm remains in the free space and does not collide with anything. However, as previously described, the free space is only a representation of the first 3 degrees of freedom of the arm. Thus, valid waypoints cannot be chosen simply by looking at the free space plot. Instead path planning for a 7 degree of freedom robot requires a bit more of a hands-on approach. In MATLAB's *Interactive Rigid Body Tree*, as shown in Figure 11, the end effector can be manually moved or rotated in any direction within the constraints of the joint angles. By doing so, viable waypoints can easily be determined for the robot to reach its destination without any collisions. The robot's goal is to perform three tasks at the wall to the right. To do so it must move from its home configuration, shown in Figure 11, to each destination at the wall. Each waypoint along the path is defined such that robot will avoid collisions and ultimately reach its destination. The path generated for this motion can be found in Section 7.2.

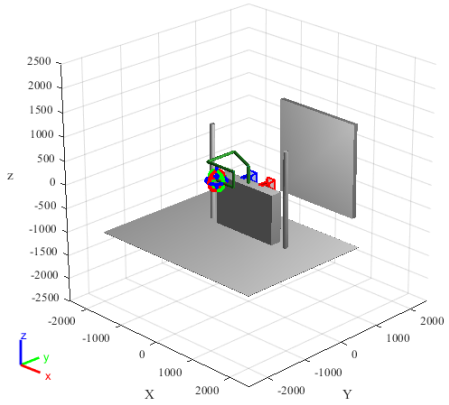

Figure 11: Interactive Rigid Body Tree: Home Configuration

The determined path is by no means the optimal path for the robot to take. It is often a choppy, jerky motion that does not result in an efficient smooth motion. The goal of path planning is not to determine the motion of the robot, but rather to determine viable waypoints the robot can visit. With these waypoints determined, steps can be taken to generate a better trajectory.

6.3 Trajectory Generation

The waypoints determined from the path planning procedure are essential to the generation of a smooth trajectory. It is now known where the robot should go in order to avoid collisions, but the question is how to get there. In order to create a smooth trajectory it is necessary to interpolate between each waypoint to generate a series of joint configurations. These joint configurations tell the robot how to move from point to point. For this project quintic interpolation methods for Cartesian motion were used to calculate the intermediate joint angles. The interpolation was performed using algorithms developed by Mathwork Inc. which generate fifth-order polynomial trajectories. These trajectories achieve a given set of input waypoints with their corresponding time values and output position, velocity, and acceleration [12]. For application in this robot, it was important to take into account velocity at each waypoint. By doing so the robot could be programmed to slow down at each turn, and come to a stop at the wall. The trajectory generated by the quintic interpolation can be found in Section 7.2.

7 Results

7.1 Vision

The vision system was successful in implementing the stereo pinhole camera solution in order to generate target locations for the robot trajectory and maintenance tasks. The error in the simulated calibrated cameras was less than 2% which for practical purposes is acceptable. This could be improved upon with

more work in the future and the introduction of real cameras into the simulation. Figure 12 shows the real points in black being observed from sample images, and then the re-projection of the estimates on top in cyan. Figure 13 shows the points in the image frame from both cameras, set into the same stereo image, including the error in each point.

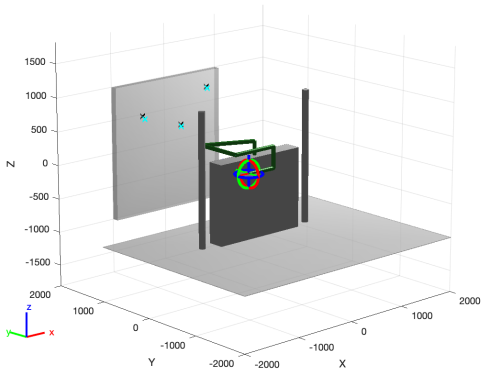

Figure 12: Camera Re-projection Estimates

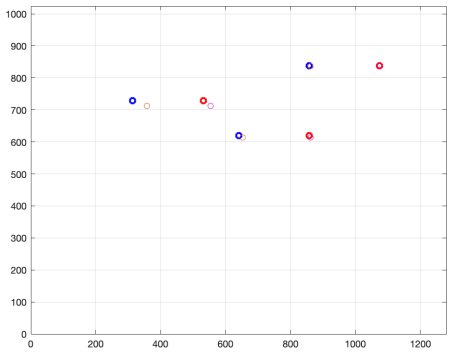

Figure 13: Camera Images With Error Estimated Points

7.2 Path Planning and Trajectory Generation

Using the process outlined in Section 6, both the path planning and trajectory generation were succesfully implemented in MATLAB. The result of the manual path planning is shown in Figure 14.

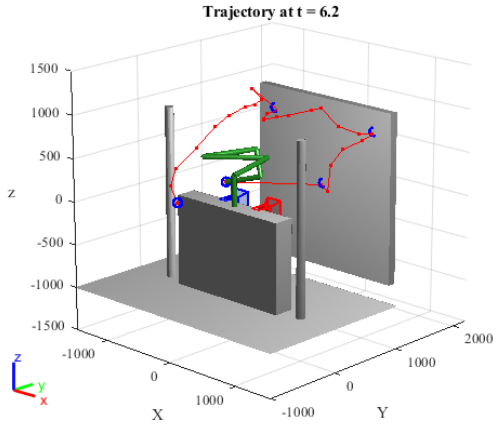

Figure 14: Path Planning Motion

As seen below, the path is quite choppy and by no means an acceptable end product. It is however, an important step in determining a proper motion for the robot. Using the waypoints determined in the path planning, the trajectory generation methods were implemented. Initially the trajectory generation was performed by using quinitic interpolation for joint space motion. This however resulted in a highly unpredictable robot motion. This was due to the fact that the intermediate joint angle configurations could be one of any infinite combinations. The result of this failed approach is shown in Figure 15.

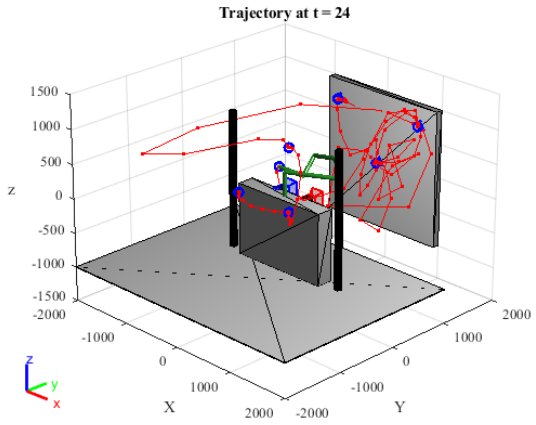

Figure 15: Failed Trajectory Generation

Given this failure, a quintic interpolation for Cartesian motion was implemented instead. This provided a much more predictable motion as the intermediate joint angle configurations were directly defined. The result of this implementation is shown in the trajectory generation plot in Figure 16.

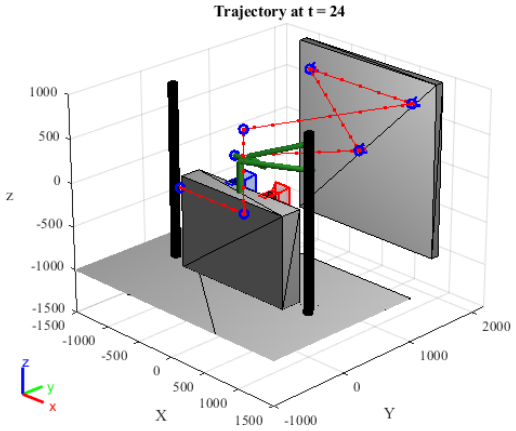

Figure 16: Trajectory Generation

In comparing the motion of the robot between the path planning and the trajectory generation, it is quickly evident the advantages that the latter presents. Rather than the inconsistent, choppy motion resulting from manual movement, the robot now moves in a smooth linear manner. This increases efficiency and reduces the risk of collision as the robot is guaranteed to visit each waypoint. With this trajectory, the motion of the robot arm was completed.

8 Future Work

While the robot is very effective in performing the tasks required, it is not without flaws. Currently a few of the joint angles at specific points along its trajectory change drastically. This could be eliminated by choosing waypoints more effectively. The optimization of waypoints also ties in to other desirable functions and elements for the robot in the future.

Currently, the path planning is done manually. Although this method is common practice in industry, this vastly limits the scope of the robot. Considering that the robot's primary application will be performing unspecified repair tasks in space, the more autonomous the system, the better suited it will be. For example, this allows the robot to be placed in many more unknown spaces. It would allow for a more versatile application range, and would enable the robot to perform effectively even in confined spaces. To do this, quadratic programming and inequality constraints will be implemented within the path planning algorithm to autonomously generate viable waypoints.

Given the most ideal case, a physical robot could be constructed to perform the tasks generated by trajectory generation. Building a physical robot could also enable further studies into end effectors. Although not addressed in the report in detail, end effectors play a significant role in the robot's functionality. Depending on the repair operation, not only would end effectors need to be modified, so would the robot pose. The simulation that is currently incorporated into this project works effectively in demonstrating rudimentary functions of the robot. Given more complex tasks such as welding and variable

plane repairs, having a physical robot becomes an invaluable resource that can be pursued.

9 Conclusion

Through the processes detailed in this report, a successful model was generated for the 7 degree of freedom robot arm for use in satellite operations. Both a robust vision and motion system were implemented and integrated in MATLAB to simulate the robot. In doing so, the framework was laid for a more in-depth study into the feasibility and automation of satellite maintenance and repair. Team 2's hope is that the findings from this project can help propel the satellite industry into the future.

References

[1] W. Xu, B. Liang, B. Li, and Y. Xu, "A universal on-orbit servicing system used in the geostationary orbit," *Advances in Space Research*, vol. 48, no. 1, p. 95–119, 2011.

[2] M.-H. T. saha, S.K, *Introduction to Robotics*, NEW Delhi, 2010. [Online]. Available: roboanalyzer.com/uploads/2/5/8/8/2588919/dhparams.pdf

[3] J. Wen, "Lecture 8 notes," 2020. [Online]. Available: https://www.overleaf.com/project/5f6b72e81d8a270001f5dc37

[4] ——, "Forward kinematics matlab examples," 2020. [Online]. Available: https://www.dropbox.com/sh/gjcd5ukxsxrqk9g/AAA15hKlbauqc5QXce4HM438a?dl=0

[5] ——, "Lecture 5 notes," 2020. [Online]. Available: https://www.overleaf.com/project/5f5d850cc363b400010d7581

[6] M. Crenganis, R. Breaz, G. Racz, and O. Bologa, "The inverse kinematics solutions of a 7 dof robotic arm using fuzzy logic," *2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2012.

[7] "Inverse kinematics." [Online]. Available: https://www.mathworks.com/help/robotics/inverse-kinematics.html

[8] I. Pylkkanen, "Quasi-newton method for optimization of complex unconstrained problems," 2020. [Online]. Available: https://www.overleaf.com/project/5fd973988fef16740e97ac33

[9] L. Moore, "Quasi-newton optimization for unconstrained problems," 2020. [Online]. Available: https://www.overleaf.com/project/5fb569432f82406c3b1ee412

[10] L. Moore, I. Pylkkanen, and S. Thirumalai, "Robotics final project." [Online]. Available: https://github.com/lachlanmoore/RoboticsFinalProject

[11] Mathworks, "Matlab imfindcircles." [Online]. Available: https://www.mathworks.com/help/images/ref/imfindcircles.html#d122e128458

[12] S. Castro, "Matlab quintic interpolation," matlab. [Online]. Available: https://github.com/mathworks-robotics/trajectory-planning-robot-manipulators

A Further Information

For an exclusive in-depth interview with the Lead Researchers at Team 2 Robotics, please visit: Team 2 Exclusive Interview

B MATLAB File Inventory

All files are hosted at the following github: lachlanmoore/RoboticsFinalProject

1. SatBotMainScript.m
2. SatBotWaypointGen.m
3. cam_def.m
4. cam_finding.m
5. cam_image.m
6. camera_robot.m
7. collisioncheck_7dof.m
8. Pinhole cam.m
9. target_def.m