Autonomous Robotic Manipulation of Deformable Linear Objects During Deep Space Maintenance and Repair Procedures

By

DIEGO ROJAS
THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Mechanical and Aerospace Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Bahram Ravani, Chair

Stephen Robinson

Stavros Vougioukas

Committee in Charge

2022

Abstract

The aim of this research is to develop the capability to sense, model, and manipulate a deformable linear object in a workspace to facilitate autonomous maintenance and repair operations in a deep space habitat. The specific capability developed in this work is the ability to clear a deformable linear object that is obstructing the path to a target object, which is to be reached. The robotic system presented herein consist of two six degree of freedom arms, working in a shared workspace, along with stereo and color cameras perceiving the environment. A perception pipeline was developed to segment a wire from a point cloud and estimate the state of the wire, directly from stereo vision. The state of the wire is represented by a set of 20 nodes, equally spaced, that when connected form the shape of the wire. To determine where to move the wire, in order to clear a path to the target object, a physics-based simulator was developed to simulate different pick and pull/push actions on the wire. The pair of actions that maximizes the space around the target object were then selected to be executed by the robot. To grasp the wire, a grasp planner was developed which solves for six degree of freedom valid grasps on the wire. The proposed system was tested on hardware and demonstrated the ability to accurately sense and estimate the position of a wire in the workspace, and move the wire to clear a path to a target object. The overall system was run 30 times, for different wire configurations, and achieved a 86.7% success rate at clearing an obstructing wire.

Acknowledgments

I would fist like to express my love and appreciation to my family for their endless support throughout my academic journey. To my mother Susana and father Jose, thank you for the sacrifices you have made which allowed me to pursue my interest and education. To my older brother Michael, thank you for being the wonderful example that you were and encouraging me to always do my best. To my younger sister Daniella, I hope this can be an example to you to follow your passions, no matter the effort.

To my advisors Professor Bahram Ravani and Professor Stephen Robinson, I would like to express my sincere gratitude for their consistent support and wonderful guidance throughout this work. I would like to thank Professor Ravani for the excellent mentorship he has provided me during my undergraduate and graduate studies, and for the many life lessons he has shared with me. Professor Ravani is an individual that I have nothing but the upmost respect for and an engineer that I strive to be someday. I will forever be grateful for my time working with him. I would also like to thank Professor Robinson for suggesting the topic of this research, and for the impact he has made on my education and in my career growth. I thank him for igniting my interest in Aerospace and for the many conversations we had discussing the future use cases for robotics in Space.

I must also express my gratitude to Professor Stavros Vougioukas, for his service on my committee and for the role he has played in my robotics education. Taking Professor Vougioukas's Agricultural Robotics course sparked my deep interest in robotics which has led me to pursue a career in the field.

Lastly, I want to show my sincere appreciation and thankfulness for my partner, Mykaela, for her endless support and encouragement throughout this journey. Thank you for motivating me to push forward when times were difficult, and celebrating my moments of accomplishment. I cannot thank you enough.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

The International Space Station (ISS), which orbits the earth, is a space station/habitat that constantly holds three to six astronaut crew at all times. Space agencies associated with the ISS are able to send and return astronauts to the ISS frequently, given that the ISS orbits in a low earth orbit. Given this, most of the functional tasks, such as maintenance and repair duties required to keep the ISS operational, are the responsibility of the astronauts stationed on the ISS at that given time. This has not been much of an issue, other than adding additional responsibilities to the astronauts, however as researchers begin to plan for deep space exploration missions where future space stations/habitats may only hold astronauts a couple of months out of the year, tasks such as maintenance and repairs must be handled by on-board autonomous systems. The concept of a Smart Habitat (SmartHab) that is capable of self-awareness and self-sufficiency through artificial intelligence (AI) and robotics is very much needed to enable missions of deep space exploration and to sustain deep space habitats that may remain inhabited for months at a time.

A space habitat presents many challenges for an autonomous robotics system in terms of performing maintenance, repair, and assembly operations [6]. A challenge of interest is the fact that a space habitat contains many subsystems which are tightly compacted in order to maximize the utilization of the volume allocated in the space habitat [7]. Moreover, these subsystems often contain deformable linear objects (DLOs) such as wires, cables, and tubes, which often obstruct the path to other components within the subsystems. Dealing with such deformable linear objects autonomously is a challenge, given their complex dynamics, deformable shape, and lack of visual features [8] [9] [10]. Moreover, subsystems often contain multiple DLOs that share the same color, making it difficult to isolate and track multiple DLOs in a workspace. This presents a challenge to the on-board robotic system that may need to reach components or areas that are obstructed by such DLOs. Having the local autonomy to identify and manipulate DLO's is essential in carrying out maintenance and repair procedures on board a space habitat with compact subsystems. Dealing

with these DLOs during a maintenance/repair task is an open research problem that merits further investigation in order to achieve autonomous self-sufficiency in a SmartHab.

1.2 Project Goals and Scope

This work focuses on the case where the robotic system must reach a target object of interest during a maintenance and repair task and a DLO is obstructing the path to that object such that a collision-free path to the target object does not exist. In this work, the DLO being considered is an electrical wire. In the scenario of interest, the robotic system must first be able to identify the wire from the environment and estimate its three-dimensional (3D) position in the environment. When no path to the target object exists, the autonomous system must then manipulate the wire in order to clear a path to the target object, while avoiding damaging the wire and colliding with other objects in the environment. Given the need to manipulate the wire and grasp the target object, this task calls for two robotic manipulators in a shared workspace, where one manipulator will clear and hold the wire for the second arm to grasp the target object.

This current research will focus on developing and demonstrating an end-to-end solution to the research problem previously described. The goal of this work is to develop sensing, planning, and manipulating capabilities to autonomously identify a DLO in the environment and manipulate it in order to clear a path to a target object. The emphasis will be on developing an autonomous system that requires minimal user input or initial conditions to perform this task.

1.3 Thesis Structure

This work begins with a background in Chapter 2 that explains in detail the concepts used in this work, as well as highlights past work in sensing and manipulation of DLOs. The goal of the background section is to inform the reader of the foundational concepts that make up this work, as well as to discuss the advantages and disadvantages found in approaches taken in past research related to the research problem addressed in this work. Chapter 3 describes the methods proposed in this work for sensing, state estimation, planning, and manipulation of a wire that obstructs a path

to a target object. The experimental results for this work are presented in Chapter 4, where results on perception and state estimation are presented, along with a total success rate for the proposed system in finding a solution to clear a path to the target object. Lastly, Chapter 5 presents the conclusion of this work, as well as a discussion of the future work that can stem from this project.

2 Background

2.1 Computer Vision

2.1.1 Morphological Operations

Morphological operations are common image processing operations applied to binary images that change the shape of the binary image [1]. To perform a morphological operation, a binary image is first convolved with a binary structuring element, then a binary output value is determined based on a thresholding result of the convolution [1]. A thresholding operation with threshold t on a binary image f is defined as [11]:

$$threshold(f,t) = \begin{cases} 1 & f \geq t \\ 0 & else \end{cases} \tag{1}$$

The structuring element, or sometimes called the kernel, is a user defined shape, housed in a matrix, that is used to probe an input image. A pixel in the structuring element, often the center, is defined as the origin and the ones in the matrix define the shape. Figure 1 shows examples of structuring elements.

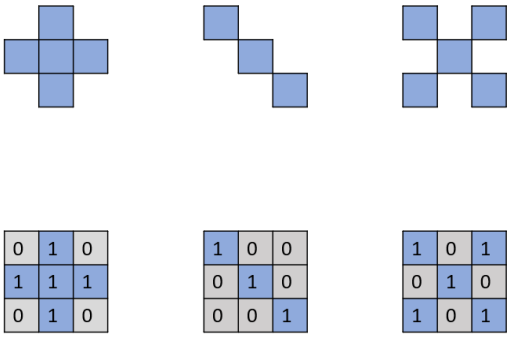


Figure 1: Examples of different structuring elements and their respective matrix implementation.

As the structuring element s acts as a moving window through the binary image f during the convolution, we define

$$c = f \circledast s \tag{2}$$

to be the number of ones in the binary image f that overlay with the structuring element s as it moves through the image and we define S to be the size of the structuring element (number of ones) in the matrix [1].

Common morphological operations are Dilation and Erosion [11]. Dilation is the process of enlarging the shape of the binary image, often expanding the boundaries of the original shape and filling in holes in the image [1]. Erosion is the process of reducing the shape of the binary image, often reducing the boundaries of the shape [1]. Applying the thresholding operation to the result of the convolution, we get the following definitions for these two morphological operations [11]:

- Dilation: dilate (f,s) = threshold (c,1)
- Erosion: erode (f,s) = threshold (c,S)

The dilation operation is defined such that if the number of ones in the binary image that overlap with the structuring element at a given position is greater than or equal to ones, then the pixel on the binary image that overlaps with the origin of the structuring element will change to a one if not already a one [1]. Whereas the definition for erosion states that if the structuring element does

not overlap completely with all ones in the binary image at a given position, then the pixel in the binary image the overlaps with the origin of the structuring element will turn into a zero [1]. Figure 2 shows the result of dilation and erosion with a 5x5 structuring element.

Figure 2: Binary image morphology: (a) original image; (b) dilation; (c) erosion [1]

2.1.2 RGB to Depth Registration

A RGB-D camera is typically made up of two sensors: an infrared sensor that captures a depth image and an RGB sensor to capture RGB color images [12]. The two sensors are placed at different locations on the body of the RGB-D camera, and therefore have different frames [12]. As a result, the pixel location of an identical object sensed in the two images differ due to the difference in frames [12]. Moreover, if the image resolutions of the two images differ, then there is not a direct one-to-one mapping of pixels between the two images [13]. To align the two images and achieve a direct one-to-one mapping, the following approach is proposed by Chaochuan [13].

Using the camera matrix model in equation 3, a 3D point $P(x, y, z)$ in the world can be mapped to a two-dimensional (2D) image coordinate $P'(u_1, v_1)$ on an image plane.

$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3}$$

where the following variable definitions are,

- (X, Y, Z) are the coordinates of the 3D point in the world frame,
- (u, v) are pixel coordinates,

- f_y, f_x are the focal lengths expressed in pixel units,
- (u_0,v_0) is the principal point that is usually at the image center.

For the RGB-D camera, each sensor has its own camera model that can map 3D world points to their 2D planar pixel coordinate system using their unique camera intrinsic parameters [13]. Equation 4 shows the relationship between the pixel coordinates $P'(u_1, v_1)$ in the depth image and the 3D point $P(X_1, Y_1, Z_1)$ in the world frame, whereas Equation 5 shows the relationship between the pixel coordinates $P'(u_2, v_2)$ in the RGB image and the 3D point $P(X_2, Y_2, Z_2)$ in the world frame [13].

$$Z_1 \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \begin{bmatrix} f_{1x} & 0 & u_{01} \\ 0 & f_{1y} & v_{01} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} \tag{4}$$

$$Z_2 \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} f_{2x} & 0 & u_{02} \\ 0 & f_{2y} & v_{02} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} \tag{5}$$

Suppose geometric pose relationship between the depth sensor and RGB sensor can be found from the camera external parameters. This can be visualized in figure [13]. Then equation 6 can be derived, which shows the transformation of 3D points in the two frames.

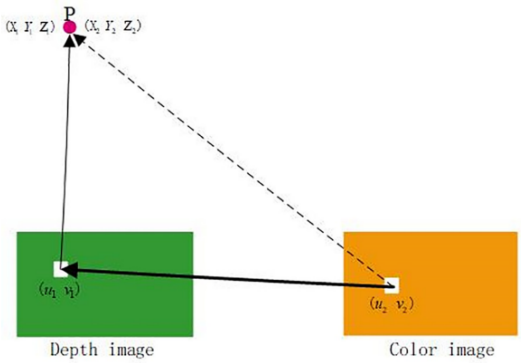


Figure 3: Visual of the depth and RGB image frame

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \\ 1 \end{bmatrix} \tag{6}$$

By taking pixel coordinates $P'(u_1, v_1)$ from the depth image, the 3D point $P(X_1, Y_1, Z_1)$ associated with those coordinates can be found with equation 4. Using equation 6, the 3D point $P(X_2, Y_2, Z_2)$, that is observed by the RGB sensor can be found. Lastly, using equation 5, the pixel coordinates $P'(u_2, v_2)$ from the depth sensor can be found, resulting in a pair of pixel coordinates from the depth and RGB image that corresponds to the same point [13].

2.1.3 K-Means Clustering

K-means clustering is a clustering technique which aims to partition a group of n data points into k clusters, where each point belongs to a cluster with the nearest mean [14]. The steps involved in the K-means clustering algorithm are as follows [14]:

1. Initialization: Randomly select k points from the set of n data points to serve as the initial set of mean values.

2. Assignment: Find the Euclidean distance from each data point to each mean value, and assign each data point to the nearest cluster mean.

3. Update: Calculate the sample mean in each cluster, and update the mean values.

4. Repeat: Continuously repeat the assignment and update steps until the assignments do not change.

The result will be k clusters in the data set, where each point in the data set will be assigned to a given cluster [14]. Figure 4 shows the pseudo code for the algorithm. To improve the clustering accuracy, the variance of each cluster is computed and recorded, and the algorithm is repeated with

a new set of *k* randomly selected mean values [2]. This process is repeated multiple times, then the solution with the lowest variance amongst the clusters is selected [2].

function Direct-k-means()

Initialize k prototypes $(w_1, \ldots, w_k)$ such that $w_j = i_l,\ j \in \{1, \ldots, k\},\ l \in \{1, \ldots, n\}$

Each cluster C_j is associated with prototype w_j

Repeat

for each input vector i_l, where $l \in \{1, \ldots, n\}$, *do*

Assign i_l to the cluster C_{j*} with nearest prototype w_{j*} (i.e., $\mid i_l - w_{j*} \mid \ \leq \ \mid i_l - w_j \mid,\ j \in \{1, \ldots, k\}$)

for each cluster C_j, where $j \in \{1, \ldots, k\}$, *do*

Update the prototype w_j to be the centroid of all samples currently in C_j, so that $w_j = \sum_{i_l \in C_j} i_l / \mid C_j \mid$

Compute the error function:

$$E = \sum_{j=1}^{k} \sum_{i_l \in C_j} \mid i_l - w_j \mid^2$$

Until E does not change significantly or cluster membership no longer changes

Figure 4: Direct K-means clustering algorithm [2]

2.2 Geometry

2.2.1 Bezier Curves

A Bezier curve is a parametric curve that uses Bernstein polynomials as the mathematical basis[15]. A Bezier curve is created from a set of *n*+1 points, where *n* signifies the order of the curve [15]. In each set of points, the beginning and end points define the start and end points of the curve, while the intermediate points are defined as the control points [15]. The control points do not generally

lie on the computed curve, but rather influence the shape of the Bezier curve [15].

The explicit definition of a Bezier for *n*+1 number of 2D or 3D points is defined in eq 7.

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i \tag{7}$$

where, parameter *t* defined as:

$$0 \leq t \leq 1$$

and where the following are the binomial coefficients:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \prod_{i=1}^{k} \frac{n+1-i}{i}$$

The first and second derivative of a Bezier curve of order n can be represented explicitly as [16]:

$$B^{'}(t) = n \sum_{i=0}^{n-1} b_{i,n-1}(t)(P_{i+1} - Pi) \tag{8}$$

$$B^{''}(t) = n(n-1) \sum_{i=0}^{n-2} b_{i,n-2}(t)(P_{i+2} - 2P_{i+1} + P_i) \tag{9}$$

where the polynomials,

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}$$

are known as the Bernstein basis polynomials [16].

2.2.2 Frenet-Serret Frame

For a continuous, differentiable curve in three dimensional Euclidean space R^3, the Frenet-Serret formulas describe the kinematic properties of a particle moving along the curve [16]. The Frenet-Serret formulas are defined for a unit speed curve $r(s)$ as:

$$\frac{dT}{ds} = \kappa N \tag{10}$$

$$\frac{dN}{ds} = -\kappa T + \tau B \tag{11}$$

$$\frac{dB}{ds} = -\tau N \tag{12}$$

where:

- T is the unit tangent vector pointing in the direction of motion,
- N is the unit normal vector,
- B is the unit binormal vector,
- κ is the curvature of the curve,
- τ is the torsion of the curve.

The tangent, normal, and binormal unit vectors collectively form the Frenet-Serret frame, or TNB frame, which together form an orthonormal basis spanning R^3 [16]. Given a non-unit speed curve $r(t)$, the Frenet-Serret frame can be found with the following equations [16]:

$$T(t) = \frac{r^{'}(t)}{||r^{'}(t)||} \tag{13}$$

$$B(t) = \frac{r^{'}(t) \times r^{''}(t)}{||r^{'}(t) \times r^{''}(t)||} \tag{14}$$

$$N(t) = B(t) \times T(t) \tag{15}$$

$$\kappa(t) = \frac{\left|\left|r^{'}(t) \times r^{''}(t)\right|\right|}{||r^{'}(t)||^{3}} \tag{16}$$

$$\tau(t) = \frac{< r^{'}(t) \times r^{''}(t), r^{'''}(t) >}{||r^{'}(t) \times r^{''}(t)||^{2}} \tag{17}$$

2.3 Simulation

2.3.1 Mass Spring Damper Model of Wire

A mass spring damper model is a common model that is used to simulate flexible objects by assuming the flexible object is made up of a collection of point masses, connected together with different classes of massless springs (linear, bending, torsional, etc.) [17] [18].With such a model, the potential energy of the system can be obtained by summing the potential energy of each spring. Due to the connected springs, the internal forces on each point mass can then be computed by taking the derivative of the potential energy $E(x)$ with respect to position of the point mass x [19]. The total force on each point mass is represented by the sum of internal and external forces, where external forces capture forces such as gravity and applied forces. The motion of each point mass can be governed by Newton's second law, shown in equation 18, which can be integrated to find position and velocity of each point mass at a given time t [19].

$$\ddot{X} = M^{-1}(-\frac{\partial E}{\partial X} + F_{external}) \tag{18}$$

This work utilizes the mass spring damper model designed by Yuan et al. (2015) [3], which models the wire as a set of point masses connected by three types of linear springs: structural springs, bending springs, and leading springs.

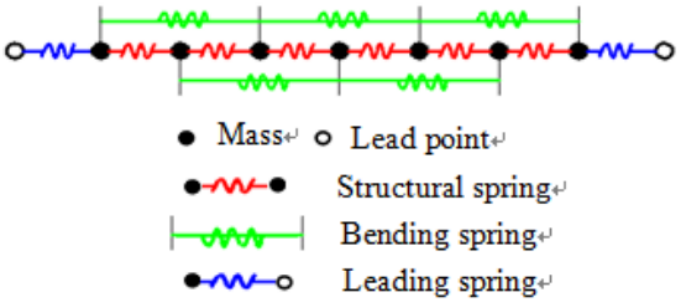


Figure 5: Mass-spring wire model [3]

Structural springs connect to each node which are useful in preserving the wire length during motion [3]. The potential energy function for structural springs in terms of the position of the point masses is defined by equation 19, where l is the natural length of the spring and k_S is the stiffness of the structural spring:

$$E_i(X) = \frac{k_S}{2}(|X_i - X_{i-1}| - l)(|X_i - X_{i-1}| - l) \tag{19}$$

The force imposed on the ith particle from the structural spring is the first derivative of the energy function in equation 19:

$$f_{Si}(X) = -\frac{\partial E}{\partial X} = k_S(\frac{l}{|X_i - X_{i-1}|} - 1)(X_i - X_{i-1}) - k_S(\frac{l}{|X_{i+1} - X_i|} - 1)(X_{i+1} - X_i) \tag{20}$$

Bending springs connect the *i*th point mass with the (*i*+2)th point mass in order to model bending characteristics [3]. The potential energy function for bending springs, in terms of the position of point masses, is defined by equation 21, where k_B is the stiffness of the bending spring:

$$E_i(X) = \frac{k_B}{2}(|X_i - X_{i-2}| - 2l)(|X_i - X_{i-2}| - 2l) \tag{21}$$

The force imposed on the *i*th particle from the structural spring is the first derivative of the energy function in equation 21:

$$f_{Bi}(X) = -\frac{\partial E}{\partial X} = k_B(\frac{l}{|X_i - X_{i-2}|} - 1)(X_i - X_{i-2}) - k_B(\frac{l}{|X_{i+2} - X_i|} - 1)(X_{i+2} - X_i) \tag{22}$$

Lastly, the leading springs are found at the ends of the wire and represent the constraints on the position of the end points due to thicker insulation at the ends [3]. The leading spring restoring force is proportional to the angle of the (*i*+1)th mass with respect to the *i*th mass, which can be visualized in Figure 6:

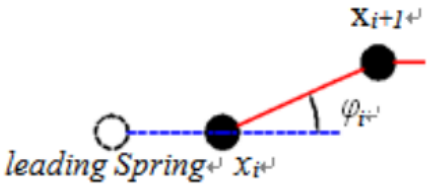


Figure 6: Leading spring [3]

The potential energy function of leading springs in terms of the position of point masses is defined by equation 23, where k_L is the stiffness of the leading spring, a_i=$(X_{i+1} - X_i)$, and b is the unit vector in the direction of the leading spring:

$$E_i(X) = \frac{k_L}{2}\phi_i^2 \tag{23}$$

$$\phi_i = \frac{k_L}{2}[tan^{-1}(\frac{|a_i \times b|}{a_i \times b})]^2 \tag{24}$$

Since the leading spring is only applied to points that are adjacent to the fixed end points, taking the first derivative of the energy function for the *i*th point that is after the fixed end results in the restoring force:

$$f_{Li}(X) = -\frac{\partial E_{i-1}}{\partial X_i} = k_L\phi_{i-1}\frac{a_{i-1} \times (a_{i-1} \times b)}{a_{i-1}^2|a_{i-1} \times b|} \tag{25}$$

whereas the restoring forces applied to the *i*th point that is before the fixed end are:

$$f_{Li}(X) = -\frac{\partial E_{i+1}}{\partial X_i} = k_L\phi_{i+1}\frac{a_{i+1} \times (a_{i+1} \times b)}{a_{i+1}^2|a_{i+1} \times b|} \tag{26}$$

With the internal and external forces computed for each point mass, Newton's second law in equation 18 is applied and solved to determine the acceleration of each particle at a given time

step. To find the position and velocity of the particle, Yuan adopted and applied the Newmark-β numerical integration method to solve the differential equation [3]. Using the Newmark-β method, the position and velocity of the ith point mass at a given time t can be found by plugging in the computed acceleration of the particles into equations 27 and 28:

$$X_t = X_{t+\Delta t} + \Delta t \dot{X}_{t+\Delta t} + \frac{1}{4}\Delta t^2(\ddot{X}_{t+\Delta t} + \ddot{X}) \tag{27}$$

$$\dot{X}_t = \dot{X}_{t+\Delta t} + \frac{1}{2}\Delta t^2(\ddot{X}_{t+\Delta t} + \ddot{X}) \tag{28}$$

2.4 Related Work

2.4.1 State Estimation for Deformable Objects by Point Registration and Dynamic Simulation

In the area of sensing and modeling linear deformable objects, the following work by Tang et al. (2017) proposes a robust state estimator to track a linear deformable objects configuration in real time [4]. The objective of this work is to use feedback from stereo cameras to estimate and track the position of a rope, characterized by a set of nodes, at each time step[4]. The approach is to combine point registration and dynamic simulation together, where point registration is used to estimate a set of nodes from a point cloud, and the dynamic simulation is used to ensure the physical constraints of the rope are respected[4]. In the point registration step, a point cloud of the rope is sampled by a Gaussian mixture model where each node used to characterize the position of the rope is regarded as a Gaussian centroid[4]. For incomplete point clouds of the rope due to occlusion, state estimation robustness is maintained by applying the coherent point drift (CPD) regularization to the mixture model, which preserves the topological structure of the rope during registration by considering state estimate in the previous time step[4]. To ensure that the estimation respects the physical constraints of the rope, a dynamic model of the rope is simulated in a physics engine where an impedance controller is used to drive the virtual rope to the shape estimated in the registration step[4]. The node positions are then sent back to the Gaussian mixture model

to initialize the registration at the next time step, allowing for tracking of the wire position over time[4]. Figure 7 shows the general architecture of this deformable object state estimator:

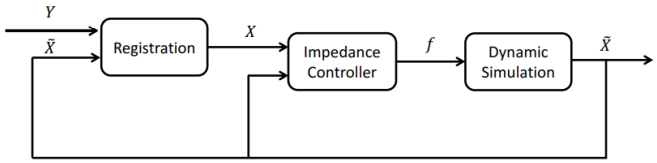


Figure 7: State estimator for tracking deformable objects [4]

Figure 8 shows the results from the Tang et al's. (2017) proposed framework, where the first column contains raw RGB images, the second column contains the point cloud of the rope, and the third column contains the state estimate of the rope. The point cloud of the wire was obtained by placing the rope in a green background and utilizing HSV based filtering to segment the rope's point cloud from the total point cloud of the environment. A one meter long rope was used in this experiment and modeled in the Bullet Physics Engine to be used in the framework. The following results shows the robustness of the proposed state estimator to sensor noise, outliers, and occlusion, which is handled in the registration step.

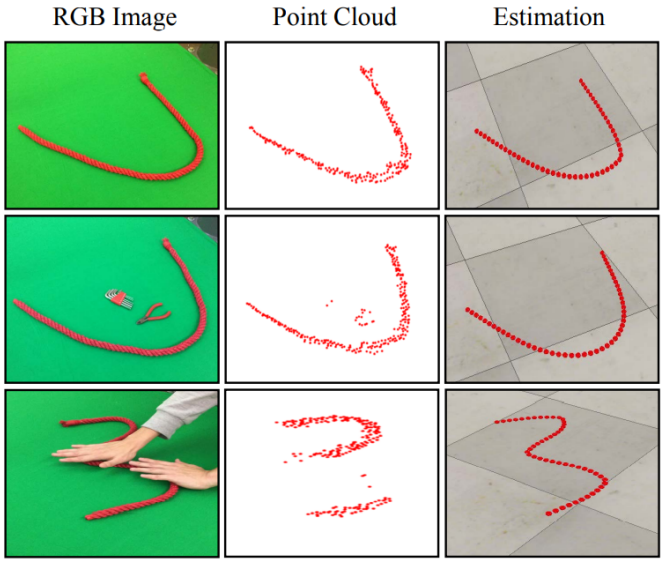


Figure 8: Results of state estimator [4]

2.4.2 A Framework for Manipulating Deformable Linear Objects by Coherent Point Drift

The work by Tang et al. (2018) presents an end-to-end framework for manipulating deformable linear objects to desired shapes autonomously [5]. This framework includes state estimation, task planning, and trajectory planning which all utilize the concept of Coherent Point Drift (CPD). Coherent Point Drift, which is a non-rigid registration method used to map on point set to another non-rigidly, is the core technique used in this work[5]. For state estimation, the rope is discretized as a chain of connected nodes with uniform distance[5]. Using CPD, the nodes of the rope at time t can be registered towards the new point cloud at time $t + 1$ in order to obtain the new state estimation of the rope at time $t + 1$ [5]. For task planning and trajectory planning, Tang et al's. (2018) work applied a method of learning from demonstrations by pre-programming robot trajectories for various rope states to create a repository of pre-recorded actions for given states. With this repository of pre-recorded rope states paired with a robot trajectory, the framework uses CPD to best match the current observed state of the rope with a pre-recorded state [5]. If a pre-recorded state that is similar to the current observed state exist, then the task planner will return the pre-recorded trajectory as the next action to be executed[5]. Since it is highly unlikely that there will be a perfect match between the current observed state and pre-recorded state, the trajectory planner utilizes CPD to create a mapping between the pre-recorded rope state and observed rope state in order to adjust the pre-recorded trajectory accordingly[5]. This is done by computing the transformation that is used to register the pre-recorded state to the current observed state, and applying that same transformation to the pre-recorded trajectory[5].

Figure 9 shows Tang et al's. (2018) results of the task and trajectory planner for the task of knotting the rope. The red lines represent the rope, the blue dots are end effector pick and place points, and the green lines are robot trajectories. Figures in (a) are pre-recorded trajectories for the given rope states, where as figures in (b) are the result from the task and trajectory planners for the observed rope states. In the figures in (b), it can be observed that the black grids represent the twisting of the Cartesian space as the pre-recorded scenarios are being mapped to the current scenarios.

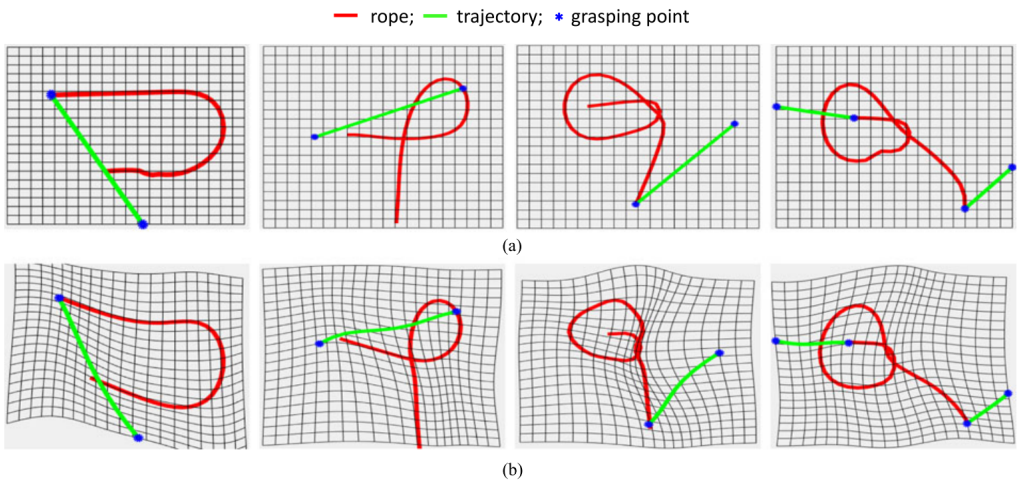


Figure 9: Results of rope knotting manipulation [5]

2.4.3 Summary

This thesis references the past work previously introduced [4] [5] for state estimation of a DLO from vision. The state estimator in [4] is highly dependent on the dynamic simulator used to ensure the physical constraints of the deformable object are respected. This method requires a complete dynamic model of the DLO beforehand, which is difficult to generalize, since deformable objects can vary in mechanical and material properties. This thesis project focuses on a method of state estimation that is not dependent on prior information such as the length, stiffness, or thickness of the DLO. Moreover, this thesis work aims to construct a dynamics model of the DLO online, directly from visual input.

Much of the past work in sensing, planning, and manipulation of DLOs have remained in a 2D tabletop setting and have not tested their proposed architectures in 3D Cartesian space [4] [5] [9] [8]. For motion planning and trajectory generation, the 2D tabletop setting simplifies the task space to movements in the x-axis and y-axis as well as rotation of the end effector about the z-axis. This thesis project focuses on tasks where the DLO spans 3D space and therefore computes motion plans for six-degrees of freedom (6DOF) end effector poses. Moreover, sensing the DLO on the tabletop setting provides a less noisy point cloud of the DLO compared to sensing it in 3D space, given that lighting conditions may differ in some areas versus others. This is especially true for

point clouds generated from stereo vision. Therefore, this master's project focuses on the sensing and state estimation of a wire that spans 3D space.

Lastly, much of the past work focuses on manipulating the DLOs to a desired state that is defined by the user. This thesis project takes a different perspective and focuses on reaching an object that is being obstructed by the DLO, therefore treating the DLO as an obstacle that needs to be moved. Based on the pose of the object of interest, the state of the DLO obtained from vision, and the state of the environment, the framework proposed in this thesis project will autonomously determine where and how to move the DLO in order to clear a path to the target object. Therefore, this thesis project attempts to increase the level of autonomy compared to past work by limiting the amount of user input required for state estimation, planning, and manipulation of a DLO.

3 Methods

3.1 System Overview

3.1.1 System Configuration

This work relies on two robotic manipulators working in a shared workspace, to move a wire that is obstructing the path to a target object. The wire is first sensed and segmented from a point cloud; the segmented point cloud is then processed and clustered to estimate the state of the wire by a set of 20 points. The set of points characterizing the wire is then used to construct and initialize a mass-spring model of the wire. With a static model of the environment and a dynamic model of the wire, a developed simulator is used to simulate different pick and pull actions on the wire in order to find the best pair of actions that results in a clear path to grasp the target object. Once a pair of pick and pull actions are determined by the simulator, methods from differential geometry are then used to plan a 6DOF pose to grasp the wire, and execute the action.

The hardware in this work consists of two ViperX 300s robot manipulators from Trossen Robotics and an Intel RealSense d435i camera mounted on a fixed camera stand. The environ-

ment mockup is designed to emulate the exterior make up of an EXPRESS Rack [20] on the ISS, which contains mounted objects and an exposed deformable object, as shown in Figure 10.

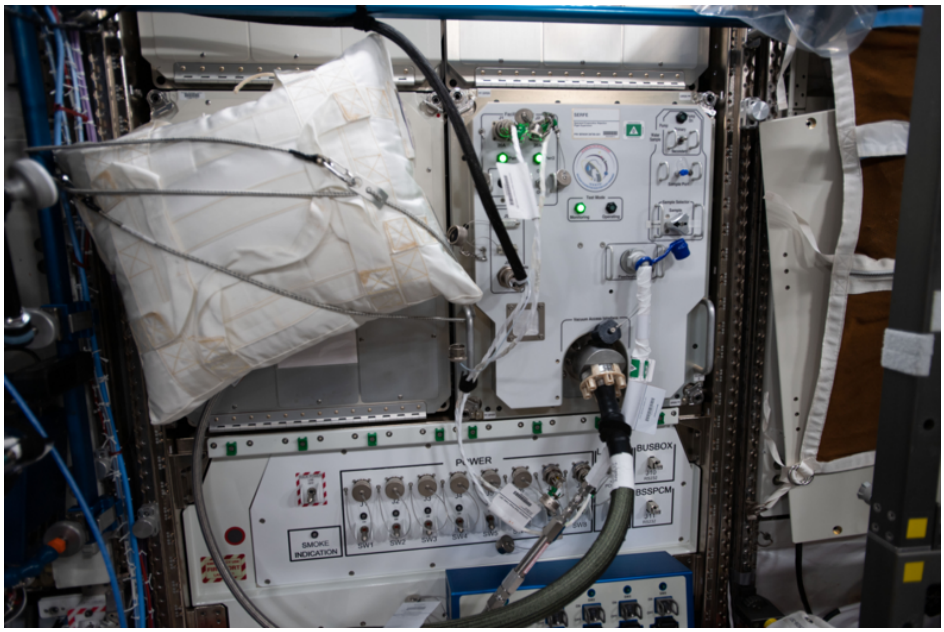


Figure 10: EXPRESS Rack on the ISS

The hardware mockup can be visualized in Figures 11 and 12, where Figure 12 displays the frames of the robot, camera, and world. The mock-up consist of the two ViperX 300s robots, offset from the world frame by seven inches along the positive and negative y-axis. The Intel RealSense d435i camera is mounted on a fixture, placing the camera -14 inches along the x-axis and +16 inches along the z-axis with respect to the world frame. Lastly, the emulated EXPRESS Rack, made with a 36 x 24 inch^2 sheet of plywood connected to a 12 x 24 inch^2 of plywood to form a right angle, is placed 20 inches along the x-axis of the world frame.

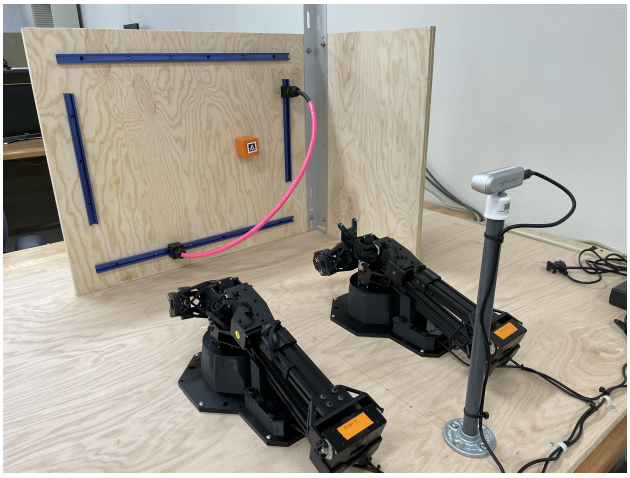


Figure 11: Hardware setup

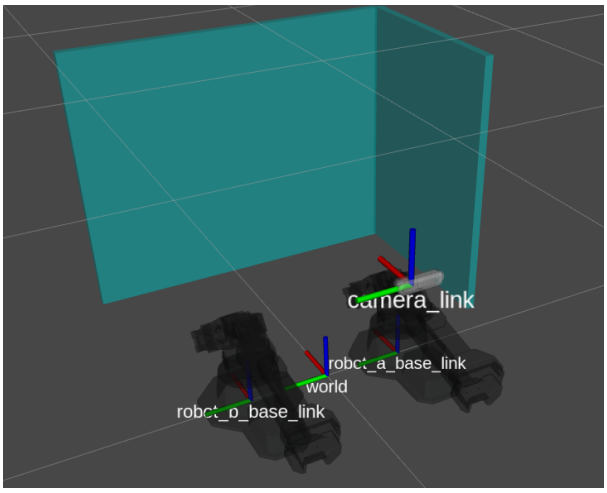


Figure 12: Rviz visual

Attached to the panel, used to emulate the EXPRESS Rack exterior, is a target object represented by the orange cube in Figure 11 and a wire that is fixed at both ends to the panel. To provide flexibility in changing the location of the fixed ends of the wire on the panel, four blue T-tracks are placed close to the border of the panel, which allow for the ends of the wire to move freely along the tracks. To accomplish this, a fixture is designed which contains an interface to the T-track at one end and a cable gland at the other which is used to secure the wire to the fixture. This allows for the ends of the wire to be placed at various locations on the four tracks, ultimately allowing for easy setup of different wire configurations.

3.1.2 Software Overview

This work was primarily developed in ROS Noetic, running on an Ubuntu 20.04 operating system. ROS is an open-source framework that provides various software libraries and tools to aid the development of complex robotic systems [21]. For this project, we leverage the messaging infrastructure that ROS provides, allowing the creation of a network of nodes that can communicate with one another through a publish and subscribe message pattern. To aid the computer vision aspects of this project, this work leverages OpenCV which is a collection of software libraries and tools that contain various computer vision functions and algorithms. The ROS framework comes equipped with a bridge to OpenCV, allowing the use of OpenCV libraries in a ROS node. For mo-

tion planning and trajectory execution, the ROS motion planning framework Moveit is leveraged and configured for a dual-arm robotic system. Lastly, custom software libraries for wire modeling and wire grasping were developed for this project.

3.1.3 System Requirements and Assumptions

The requirements of the system are as follows:

1. The vision system will demonstrate the ability to segment a wire from a 3D point cloud.
2. The state estimation layer will demonstrate the ability to estimate the position of the wire directly from the point cloud of the wire, with no prior information pertaining to the wire.
3. The simulation will demonstrate the ability to plan for a pair of actions which will successfully move the wire such that a path is cleared to the target object.
4. The time taken to run the simulation and compute the pair of actions must **not** take more that 3 seconds.
5. The system will demonstrate an ability to direct tasks to either robot in order to avoid excess crossing of the arms in the shared work space.

Assumptions:

- The pose of the target object is known from a world model.
- The geometry of the panel and ground is known to the robot.
- The system has no prior knowledge of the wire length or thickness.
- The robots have the ability to move their base to a position that is in front of the panel.
- The two ends of the wire are fixed to the panel.
- Only the ends of the wire are fixed and the rest of the wire is free.
- The procedure to insert or remove the target object is known by the system.

3.2 Segmentation

To manipulate a wire to clear a collision free path to a target object or location, the 3D state of the wire must be known. In an autonomous system that is unaware of the state of the wire, sensory information can be used to identify and estimate the state of the wire in the environment. For this work, the sensory information is coming from a RGB-D sensor, where concepts from computer vision are then leveraged to segment the wire from a point cloud of the environment.

A point cloud obtained by RGB-D camera is inevitably noisy compared to other point cloud sensors, such as LiDAR. Such noise is due to outliers that do not belong to the surface of objects, due to effects from light intensities, different viewing angles, and reflection of the object. As a result, this work focuses on segmentation of a wire in a 2D RGB and depth image, then converting the segmented depth image to a point cloud in order to obtain an isolated point cloud of the wire.

3.2.1 Segmentation of Wire in 2D

Segmentation of a wire in a 2D image was not a main focus in this work, given the vast amounts of methods for 2D segmentation. Rather, the focus of this work was accurately segmenting a wire from a 3D point cloud by using a 2D image. Given this, the segmentation of a wire in a 2D image is achieved by filtering the wire from the background based on the Hue Saturation Value (HSV) of the wire [22]. To clearly differentiate the wire from the background by color, a pink wire was chosen for this work. This segmentation technique can be replaced in the future with more robust techniques such as [11] [10]. To perform the segmentation technique based on HSV color filtering, the following steps are taken:

1. Pass RGB image to the *cvtColor* OpenCV function to convert from RGB to HSV color model
2. Use the OpenCV *inRange()* function to create a mask of all the pixels that fall within the specified color range
3. Apply the bitwise AND operation to the raw image using the masks from step two to segment the original image.

4. Convert to binary image

The output from HSV filtering is a binary image, showcasing a roughly segmented wire with expected noise from the background. To eliminate this noise and further segment the wire from the background, the following operations are applied on the binary image:

1. Dilation,

2. Contour Detection: Finding the largest contour,

3. Erosion.

Dilation is performed to enlarge the boundary of the wire in the segmented binary image. This is an important image processing step in segmentation, given that the output from raw segmentation may contain holes in the object, missing boundaries, or disconnected portions of the object. Dilation is a useful step to fill in holes and gaps in the raw segmented binary image, to better capture the object in the image. To execute this, a 3x3 structuring element, shown in Figure 13 was selected and used with the openCV *dilate()* function.

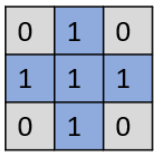


Figure 13: Structuring element

In the raw segmented image, other unwanted pixels that do not belong to the wire object may be picked up as outliers. This is especially the case when segmenting based on HSV values, given that other objects in the background may be similar in color. To extract only the pixels associated with the wire, the OpenCV *findContours()* function is applied to the dilated binary image which outputs a list of contours found in the image. Using the *contourArea()* function, the list of contours is swept through to compute and extract the contour with the largest area. Given that this procedure is applied on the previously segmented image, the contour with the largest area

inherently corresponds to the wire object and therefore filters out pixels not associated with the wire object. Dilation on the binary image is an important pre-processing step to contour detection in order to close any gaps or holes in the segmented wire object. Doing so creates a single contour that represents the complete wire object, as opposed to multiple contours which may result from gaps and holes in the original segmented image.

Figure 14 showcases the outcome of these steps, where the first image is the output from raw segmentation, the second image is the output of dilation on the segmented image, and the third image is the largest contour found in the dilated image. It is apparent that the dilation step closes some gaps and missing boundaries in the segmented image, as well as exposes some outliers in the image. These outliers are filters in the contour detection where only the contour with the largest area is kept.

(a) (b) (c)

Figure 14: (a) Raw segmentation, (b) Dilation, (c) Largest contour extraction

Lastly, the erosion operation is performed on the binary image output from the contour detection step. This is done using the same structuring element as shown in Figure 13 and the *erode()* OpenCV function. The purpose of applying the erosion operation is to counteract the dilation step and eliminate the added boundary to the wire. The overall output is a binary image where ones represent the pixels associated with the wire, and zeros represent everything else.

3.2.2 Segmentation of wire in 3D

By simultaneously capturing an RGB and depth image with the Intel RealSense d435i, the output from 2D segmentation can then be used as a binary mask to segment the wire from the corresponding depth image. However, there is a difference in frame resolution between the RGB image and depth image, where the RGB image has a frame resolution of 1920 x 1080 pixels, whereas the depth image has a resolution of 1280 x 720 pixels. Given the RGB image resolution differs from the resolution of the depth image, there initially is not a direct one-to-one mapping of pixels between the two images. Therefore, prior to filtering the depth image with the RGB image, a depth to RGB image registration step is required to align the depth image to the RGB image such that there is a direct one-to-one mapping of pixels.

Upon completion of the depth to RGB registration step, the RGB image is then iterated through in search for the pixels containing ones, which results in pixels associated with the wire object. For every pixel at row v and column u in the RGB image that contains a zero, the depth value at row v and column u in the depth image is set to zero. Otherwise, if the row v and column u in the RGB image that contains a one, the value in row v and column u in the depth image is untouched. This results in a modified depth image that only contains depth values in the pixels that make up the wire object.

With the now segmented depth image that contains depth values associated with the wire, the depth image is then converted into a point cloud. To do this, the *depth image proc* ROS package from the *image pipeline* is leverage which contains a ROS node that takes in an ROS Image data structure and outputs a point cloud of type PointCloud2. The outcome of this is a point cloud consisting of points that make up the wire, resulting in segmenting a wire from a 3D point cloud. Figure 15 shows the result of this process where the raw RGB image is shown in the bottom left, the segmented depth image is show in the top left, and the segmented point cloud is shown on the right.

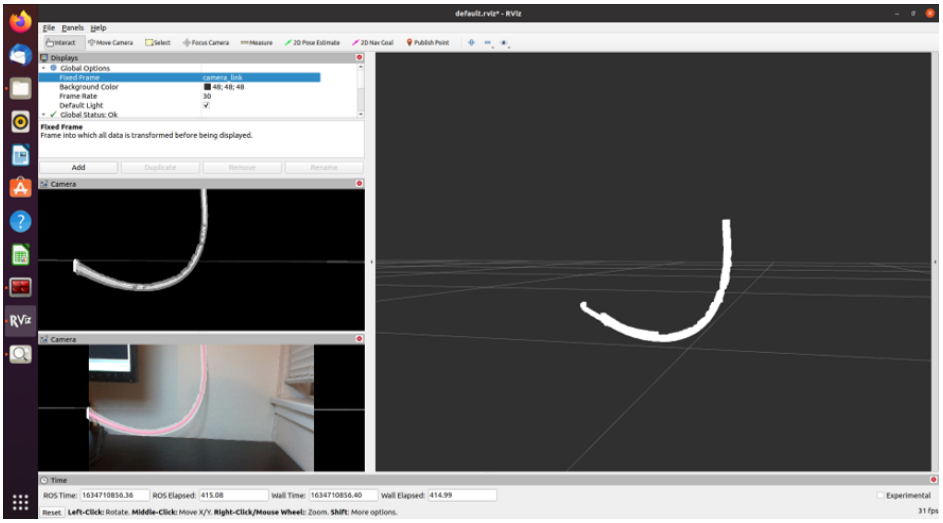


Figure 15: Segmentation of a wire from a 3D point cloud

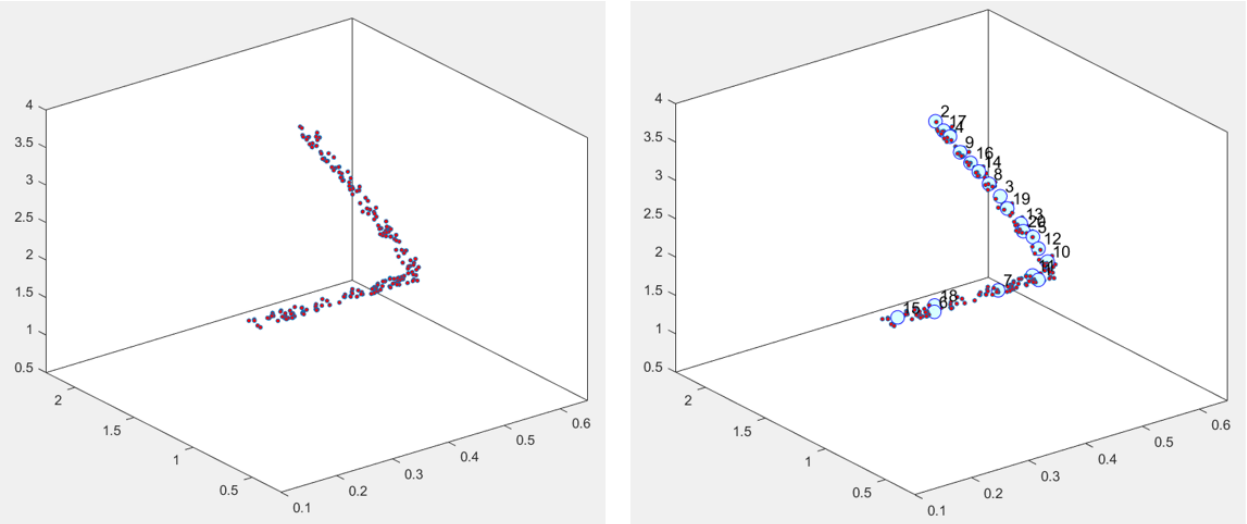
3.3 State Estimation

3.3.1 Clustering and Filtering

Deformable linear objects, such as a wire or rope, can be described and modeled as a chain of connected nodes [4]. The goal of this portion of the work is to convert the segmented point cloud sensed from a depth camera directly into a set of 20 connected nodes which describe the wire. This is done without prior knowledge of the wire characteristics, such as length or thickness.

The point cloud of the wire contains roughly 200 - 400 points, after segmentation. The challenge here is to quickly and efficiently generate a correspondence between a set of 20 connected nodes and the point cloud of the wire. For this, K-means clustering is first applied on the point cloud to simplify the 200 - 400 points down to a set of 20 mean value clusters. K-means clustering on the point cloud is effective here because it can quickly generate a solution that represents the makeup of point cloud with a simplified subset of points. These 20 mean value clusters serve as an initial guess at characterizing the wire with a set of connected nodes. Figure 16 shows the outcome of applying K-means to a point cloud. Although K-means gives a fast simplification of the point cloud and initial estimation of the chain of points, the output array of points from K-means does not come in a sorted order that would follow the shape of the wire. This is inherently due to

the nature of the algorithm to start off with a random selection of mean value clusters in order to improve computation time. This means that the nodes adjacent to each other in the output array are not guaranteed to be adjacent to each other in the Cartesian space. This can be seen from Figure 16 in the clustered image, which shows a number next to the points, indicating where that point lives in the output array. Therefore, the output array must be sorted such that the order of the connecting nodes that make up the wire in the Cartesian space is reflected in the array.



(a) (b)

Figure 16: (a) Raw point cloud (b) Clustered point cloud

Prior to sorting the output array, potential outliers are first checked and removed. For this, a custom algorithm was created that eliminates points from the set of 20 points outputted from the K-means algorithm based on the Euclidean distance to other points. This was inspired from the idea that in a set of *n* connected points, the points should ideally be spaced relatively close to each other. Therefore a maximum distance d is defined as the maximum distance a point should be spaced from a neighboring point, to be considered in the connected chain. For a point i to remain in the initial array of n points, there must be at least one point j in the array, such that the Euclidean distance between i and j is less than or equal to d. Satisfying this condition means that the point has an adequate neighbor and likely lies in the chain of points that makes up the wire. Failure

to meet this condition means that the point does not have an adequate neighbor and is likely an outlier, and this results in it being removed from the set. The pseudo code for outlier detection is shown in Algorithm 1:

Algorithm 1 Outlier Detection and Removal

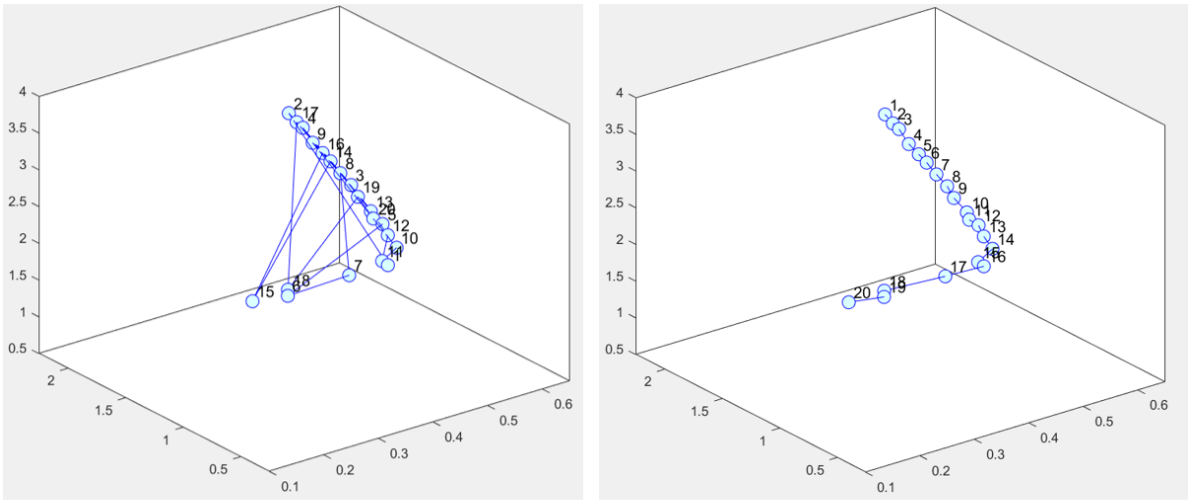
```
for i = 1, 2, ..., N do
    for j = 1, 2, ..., N do
        if i ≠ j then
            Find Distance between point i and j
            if dist ≤ mindist then
                mindist = dist
            end if
        end if
    end for
    if mindist ≥ maxdistance then
        remove outlier
    end if
end for
```

With outliers removed, the remaining points from the initial set must then be sorted to create a chain of connected points. Common trends for sorting data are sorting an array in descending or ascending order. However, given that the shape of a deformable linear object is highly non-linear, there is not a consistent trend in data that can be used to sort these set of points such that the order corresponds to the order found in chain of points that make up the wire. To sort this randomly ordered set of points, influence is drawn from path planning and graph-based algorithms [23] [24] to find the set of ordered points that result in the shortest Euclidean path. A path length can be computed by summing the distances between point i and point $i + 1$ where i = 1,2 ... n. Therefore, the shortest path is the case when the order of the points in the path makes a chain of connected points.

To begin this sorting procedure, the starting point must first be identified. The starting point in this case corresponds to one of the ends of the wire, and the ending point is the other end of the wire. Therefore, the point in the data set that corresponds to one of the end points of the wire must be found. With the assumption that the wire ends are fixed on the panel in front of the robot, and

that the panel lies parallel to the y-z plane in the world frame, the points with the maximum and minimum values in the y and z-axis are identified in the set. This results in four values, a max and min value for both the y and z-axis. The difference between the maximum and minimum values in both y and z are then computed and compared against each other. If the difference in y is greater than the difference in z, then the wire is predicted to mostly span the panel horizontally and the end points of the wire are taken to be the points that contain the maximum and minimum y-values. Alternatively, if the difference in z is greater than the difference in y, then the wire is predicted to span the panel mostly vertically, and therefore the end points of the wire correspond to the points containing the maximum and minimum z values.

With the end points identified, the points can then be sorted to create the chain of nodes. Taking one the previously determined end point in the unsorted array, a new array is created and the end point is stored in the first element of the new array. This new array will make up the sorted chain of nodes. Starting with the end point i in the new array, the unsorted array is iterated through in search for a point which minimizes the Euclidean distance to i. This point will be marked as point $i + 1$ in the chain of nodes, and appended to the new array. This processes is repeated with the next point until all elements in the new array are populated. The result is a sorted set of points that make up a chain of connected points that describe the wire. This can be visualized in Figure 17.



(a) (b)

Figure 17: (a) Unsorted (b) Sorted

3.3.2 Curve Reconstruction

Given the nature of the K-means algorithm to shift mean values towards condensed areas in the point cloud data, it is often the case that these mean values, described by an *XYZ* coordinate, are not uniformly spaced with respect to neighboring mean values. This is often the case in the event where there is occlusion in the point cloud, causing disconnected segments. In this case, the wire must be reconstructed such that the chain of points characterizes the full wire shape and have uniform spacing relative to each other. To do so, a 3D Bezier curve made up of 100 points is constructed by using the sorted points initially obtained from K-means as the control points for the curve.

To compute the Bezier curve, a python wire modeling library was developed using the mathematical formulations presented in section 2.2.1. In addition, the library contains methods to compute the first and second order derivative of the Bezier curve for a set of n points based on the formulation presented in section 2.2.1. Figure 18 shows the result of a Bezier curve from the set sorted points, initially output from K-means. Visually it can be observed that the Bezier curve tracks the shape of the point cloud quite well. From the generated Bezier curve, made up of 100 points, 20 points that are spaced five elements apart in the array are extracted to make up the new complete chain of points to characterize the wire. This can be seen from Figure 19.

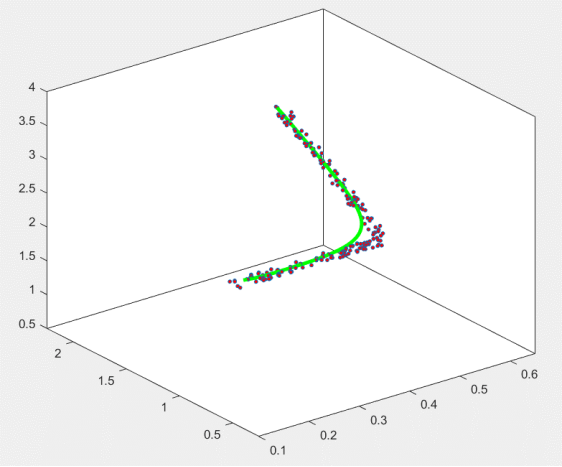


Figure 18: Fitting Bezier curve to data

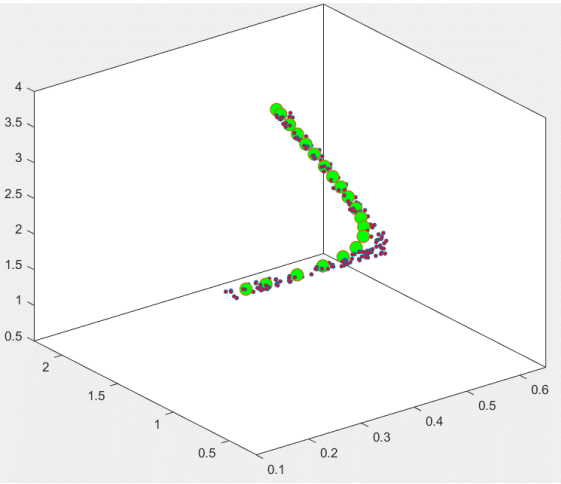


Figure 19: Final chain of nodes

Using a Bezier curve to represents the wire state comes with much upside. To begin with, for cases where the point cloud is affected by occlusion, fitting a Bezier curve to the mean clusters allows for a complete construction of the wire by bridging gaps that are present in the point cloud as a result of occlusion. In addition, a Bezier curve is smooth in nature, which makes for a more natural representation of the wire. This is important in the next section when these sets of nodes are used to initialize a wire simulation model used for planning. Last, having an equation for a curve that represents the wire allows for the properties of curves to be leveraged, such as computing the curvature or tangent vector which will be used in the wire grasp planning stage of this work.

3.4 Task Planning with Simulation

In this section, the method of determining robot actions to move the wire to clear a path to the object is explained. Two high level robot tasks to move the wire are determined using a developed simulator which contains a dynamical model of the wire, collision objects in the environment, and the target object of interest. The first task is to find the best place on the wire for the robot to grasp, and the second task involves determining where to move the wire such that a second arm can reach in and grasp the target object that was being blocked by the wire. Using the previously determined chain of 20 points that estimate the position of the wire and known pose and geometry information of objects in the environment, the simulator simulates multiple pick and push/pull actions in order to predict and determine the best set of actions to move the wire.

3.4.1 Wire Simulation Overview

A lightweight simulation was custom made to for this project with the intention to be used within the control loop. The simulation involves a dynamics model of the wire described by a chain of nodes, a static 3D model of the target object, and static 3D representations of collision objects in the environment. The wire model was derived from [3] which was reviewed in the background section of this paper. For this project, only the structural springs and bending springs were used in the wire model. The lead springs did not make a significant impact for this application, and

therefore were neglected to increase computational speed.

The wire thicknesses of interest in this thesis project are between 0.25 - 0.5 inches, which covers a variety of common wire types, as well as wire harnesses made up of thinner wires. The dynamics model used in the simulation was modeled around properties of a wire with a thickness of 0.375in and a mass of 0.037kg per meter, which falls in the center of the range of wire thicknesses being considered in this project. Selecting a wire model that falls in the middle of the wire thickness range considered allows for the simulator to be applied to multiple wire thicknesses with an appropriate level of confidence that the model will not diverge too far from the physics of the real wire currently considered. It is important to note that this application does not call for a high fidelity model of the wire, but rather calls for a low cost model that can predict roughly where the wire will be in space when a force is applied to a point on the wire, in a certain direction. The reason for this is that the focus of this work is to not manipulate the wire to a desired shape, but rather to determine where to move the wire such that a path is cleared to a target object. Therefore, an approximate model of the wire is appropriate for use in this case.

The natural length of the wire L is computed from the vision pipeline by summing the Euclidean distance between the nodes in the chain. The spacing between each node, represented by l, is then the total length L divided by $N-1$ nodes. The length l is used to represent the natural length of the springs used in the wire model, where the natural length of a structural spring is l and the natural length of a bending spring is $2l$. The total length L of the wire is used to determine the total mass M of the wire by multiplying the length L and the constant 0.037kg/m. With a chain of 20 nodes making up the wire, the total mass m for a single node was set to be $M/20$. Lastly, the stiffness constants used in the wire model are shown below. These values were motivated by [3] and ultimately tuned in order to stabilize the simulation model.

k_s=260N/m

k_b=122N/m

For each node i, in the chain of nodes making up the wire, where i = 1,2,3...20, the wire model

was set up as follows, where X is the position vector of the node i:

For nodes i = 2-19, the structural spring force on each node i is:

$$f_{Si}(X) = k_S(\frac{l}{|X_i - X_{i-1}|} - 1)(X_i - X_{i-1}) - k_s(\frac{l}{|X_{i+1} - X_i|} - 1)(X_{i+1} - X_i) \tag{29}$$

For nodes i = 3 - 18, the bending spring force on each node i is:

$$f_{Bi}(X) = k_B(\frac{l}{|X_i - X_{i-2}|} - 1)(X_i - X_{i-2}) - k_B(\frac{l}{|X_{i+2} - X_i|} - 1)(X_{i+2} - X_i) \tag{30}$$

For node i = 2, the bending spring force is:

$$f_{Bi}(X) = -k_B(\frac{l}{|X_{i+2} - X_i|} - 1)(X_{i+2} - X_i) \tag{31}$$

Where as for node i = 19, the bending spring force is:

$$f_{Bi}(X) = k_B(\frac{l}{|X_i - X_{i-2}|} - 1)(X_i - X_{i-2}) \tag{32}$$

Nodes i = 1 and 20 are treated as fixed end points and therefore their acceleration and velocity are zero and their position does not change. The states for nodes i = 1 and 20 are shown below.

$$\ddot{X}_i = 0$$

$$\dot{X}_i = 0$$

$$X_i = X_f$$

For the remaining nodes, the acceleration for each node is found by summing the internal and external forces, and dividing the sum of forces by the mass of the node represented in equation 33 and more explicitly is equation 34

$$\ddot{X}_i = \frac{1}{m}(f_s + f_b + f_{gravity} + f_{applied}) \tag{33}$$

$$\ddot{X}_i = \frac{1}{m}\begin{bmatrix} f_{s_x} \\ f_{s_y} \\ f_{s_z} \end{bmatrix} + \frac{1}{m}\begin{bmatrix} f_{b_x} \\ f_{b_y} \\ f_{b_z} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m}\begin{bmatrix} f_{x_{applied}} \\ f_{y_{applied}} \\ f_{z_{applied}} \end{bmatrix} \tag{34}$$

3.4.2 Target Object

Along with the wire model, the aspects of the target object are included in the simulation. The main components needed to model the target object in the simulator are:

- Pose of the target object
- Dimensions of the target object
- Grasp axis to insert/remove target object

It is assumed that the above information is known beforehand from a world model. The pose of the target object is taken with respect to the world frame, much like everything else. The dimensions of the target object are estimated by a rectangular prism that encapsulates the objects volume, with some extra buffer. This dimension definition will become more important when defining volumetric constraints. Lastly, the grasp axis is defined as the axis, in the target object's local coordinate frame, where the part needs to be inserted/removed along. To best describe this, the bolt in Figure 20 was used as an example for a target object.

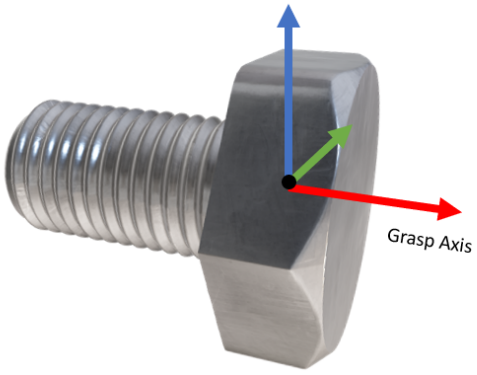


Figure 20: Frame used for target object

Much like many components in a subsystem that need to be either assembled or disassembled, the bolt has a specific and unique direction at which it must be inserted into or removed from another component. Given this, this work defines that direction as the "grasp axis". In Figure 20, in the bolt's local frame, the grasp axis is defined as the $+x$ axis. By convention, this defines the axis and direction to 'remove' the target object. To insert the bolt, it would simply be the performed in the opposite direction. This parameter is important when determining the best place to move the wire to clear a path given that the motion at which the robot must remove the object is constrained. Therefore, the volume needed to remove this object must be known and considered when determining the best place to move the wire. This will be explained in greater detail in the upcoming sections.

3.4.3 Grasp Point and Pull Vector Generation

When grasping a rigid object with a parallel jaw gripper, the selection of a grasp pose must carefully consider the geometry of the rigid object in order to find a antipodal grasp. However, for a deformable linear object such as a wire however, given the uniformity of the wire, there are multiple valid end effector poses to grasp the wire at any given point on the wire. Therefore, the solution space to find a valid grasp for the wire is quite large. The challenge for this work however is to find the best point on the wire to grasp so that the wire can be moved to create the largest path to the target object. For a wire that is fixed at both ends, the point on the wire that will allow for the greatest motion is at the center of the wire. This is generally a good selection for a grasp point for our application. However, for the case where the target object is offset from the center of the wire, selecting a grasp point at the center of the wire may not be the best option, given that only the volume near the center of the wire is being manipulated and not the volume directly associated with the target object. Rather, a better point to grasp on the wire would be closer to the object so that moving the wire at that point will result in manipulating more of the volume that is closer to the target object.

To determine the grasp point generally, we must find the point on the wire that best overlaps

with the center of the grasp object. To do this, the nodes that make up the wire are projected onto the plane of the target object, and the closest node to the target objects origin is selected as the grasp point. The plane of the target object is represented as the y-z plane of the target objects local frame, which can be visualized in Figure 20. The x-axis of the target objects local frame represents the normal vector $\vec{n}$ of this plane and the origin of the target object local frame is P_{target}. The origin of this frame and x-axis are known from a world model, as explained in the previous section. Finding the position vector $\vec{r}$ from the target object origin P_{target}, to a single node P_{node} in the chain that represents the wire, equation 36 can be used to find the projection of the node P_{node} onto the target object plane. This is performed for all nodes in the wire model, which results in a new set of points which are projected onto the plane of the target object. With this set of points that lie on the target object plane, the Euclidean distance from each projected point and the origin of the target object is computed and stored. The node which corresponds to the closest projection to the origin of the target object is then selected as the grasp point.

$$\vec{r} = \vec{P_{node}} - \vec{P_{target}} \tag{35}$$

$$\vec{Proj_i} = \vec{P_{node}} - (\vec{r} \cdot \vec{n}) * \vec{n} \tag{36}$$

When grasping and moving a point on a wire that is fixed at both ends, the motion is restricted to move along the normal of the wire at that given point. Motion along the wire is minimal, given that the wire is fixed at both ends. Therefore, the motion planning for the wire is focused on the plane spanned by the vectors normal to the wire where the grasp point resides. To find this plane, the Frenet-Serret frame was found at a given point on the wire, which describes the tangent, normal, and bi-normal vectors at a point on the curve. The computation of the tangent, normal, and bi-normal frame is done by finding the first and second derivative of the Bezier curve $b(t)$ and implementing equations 37, 38, 39. The generation of this frame is available as a method of the custom python wire modeling library developed for this work:

$$T(t) = \frac{b'(t)}{||b'(t)||} \tag{37}$$

$$B(t) = \frac{b'(t) \times b''(t)}{||b'(t) \times b''(t)||} \tag{38}$$

$$N(t) = B(t) \times T(t) \tag{39}$$

Computing the normal and bi-normal vectors at the grasp point, the plane made by these two normal vectors is used to determine six more normal vectors. A total of eight normal vectors are defined as candidate pull/push vectors to move the wire. Figure 21 shows a visual of these eight vectors:

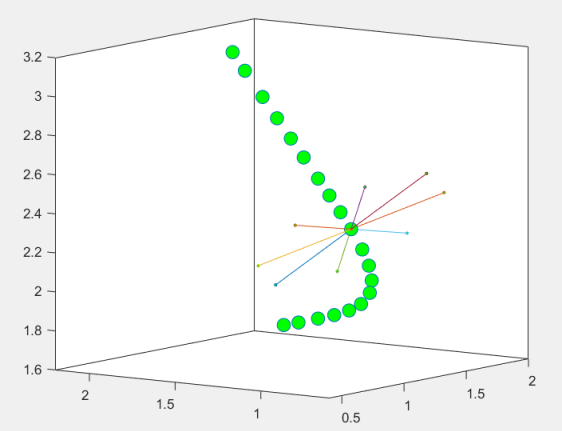


Figure 21: Normal vectors

3.4.4 Wire Simulation

With the grasp point and eight different pull/push vectors determined, the simulator is used to determine the end wire configurations when a force is applied to the grasp point along the eight different push/pull direction vectors. The magnitude of the applied force $F_{applied}$ was selected to be $2N$, which is roughly a third of the maximum lifting force of the ViperX 300s, and is therefore reasonable for this use case. For each pull/push vector, a corresponding force vector is calculated by multiplying the push/pull vector by the magnitude of the applied force.

$$\vec{F_i} = F_{applied} * \vec{V_i} \tag{40}$$

With eight calculated applied force vectors, the simulation is run eight times to determine the

different wire configurations as a result of the applied force. Newton's second law presented in equation 34 is used to generate a differential equation for the acceleration of each node from the sum of the total force experienced by each node. Using the Newmark-Beta method of numerical integration presented in equations 27 and 28, the differential equation can be solved to determine the velocity and position of each node at each time step. For this simulator, a time step of 0.001s was selected which provides fast computation speed without jeopardizing accuracy. The simulation terminates when one of two conditions are met: 1) The velocity of each node $\approx$ zero or 2) The number of iterations reaches 1,000. The pseudo code for the simulation is shown in Algorithm 2, below:

Algorithm 2 Simulation

```
for j = 1, 2, ..., 8 do
    Initialize position of nodes in wire model
    Get applied force vector F_applied
    while Condition = True do
        Compute total forces on nodes
        Find position and velocity of nodes at t = i+1
        if node velocity ≈ 0 or iterations ≥ 1,000 then
            While loop Condition = False
        end if
    end while
    Store final node positions
end for
```

The output of the simulation, after running eight times for the different pair of actions, is a 3x20x8 tensor P_{wire} that holds the eight different wire configurations, each described by a 3x20 matrix P_{nodes}. The columns of P_{nodes} hold the 3D position of each node in the chain that models the wire. To increase the speed of the simulation, elements are organized in matrices form to leverage fast matrix operations. Shown below are the different elements involved in the simulation.

$$V_{pull/push} = \left[M_{3\times 8} \right] \text{ - Pull/Push vectors}$$

$$F_{applied} = \left[M_{3\times 8} \right] \text{ - Force vectors}$$

$$P_{nodes} = \left[M_{3\times 20} \right] \text{ - Chain of nodes position}$$

$$v_{nodes} = \left[M_{3\times 20} \right] \text{ - Chain of nodes velocity}$$

$$a_{nodes} = \left[M_{3\times 20} \right] \text{ - Chain of nodes acceleration}$$

$$F_{nodes} = \left[M_{3\times 20} \right] \text{ - Total forces on chain of nodes}$$

$$P_{wire} = \left[M_{3\times 20\times 8} \right] \text{ - Holds the 8 sets of wire positions}$$

3.4.5 Grasp Point and Pull Vector Selection

After simulating different actions on the wire and generating the eight wire configurations as a result of these actions, the results from the simulator stored in P_{wire} are then processed to select the best pair of pick and pull/push actions to clear a path to the target object. The first step in the selection process is to check for any collisions between the eight wire configurations and the environment. If any node in a wire model collided with a defined collision object in the environment, then the pair of pick and pull/push actions that produced that wire configuration were removed from the set of potential solutions.

To enforce collision checking, the Flexible Collision Library (FCL) [25] was leveraged in the simulation framework. At a minimum, the collision objects of interest were the panel that held the wire, the floor, the target object, and the 20 nodes that make up the wire model. More collision objects can easily be included for more complex environments. To implement collision checking in the simulator, a custom class was created using FCL's class methods in order to build primitive collision objects and store them in a list for easy reference. For this work, only boxes and sphere collision objects were implemented for simplicity, however this class can be extended for more complex geometries. To make a collision object, the user must call either the *make_box()* or the *make_sphere()* method and pass in the dimensions and position of the object as arguments. On the back-end, these parameters are used to define an FCL collision object and appended to a list in the class.

Since the goal is to check for collisions between the environment and the nodes in the wire model, the collision objects are divided into two groups: 1) Environment object and 2) Wire model. Therefore, two instantiations of the collision object class are made for each of the two groups which results in two separate lists of collision objects. By leveraging FCL's octree data structure and broadphase collision checking, collision checking between the two groups can be achieved very efficiently. This is much superior to pair wise collision checking, where objects are treated individually and are checked against other collision objects in space, which introduces n-squared complexity.

After checking for collisions, the set of wire configurations that do not collide with the objects in the environment remain. From this sub-set, the goal is to converge on a single wire configuration that allows the clearest path to the target object so that a robot arm can reach in, grasp, and remove it. Therefore, the clear path must be large enough such that the size of the target object and robot end effector can pass through free of collisions. Given the constraint that the target object must be removed or inserted along its grasp axis, this problem is formulated to find a clear path such that the boundaries of the target object and end effector can travel along the grasp axis of the target object without colliding with the wire. This formulation can be evaluated in 2D, given the constraint on the motion of the target object, by projecting the geometry of the target object, end effector, and the wire model nodes onto the plane normal to the grasp axis of the target object. The projected nodes on this Cartesian plane that overlap with the area of the target object would ultimately collide with the target object if it were to travel along the grasp axis in 3D space. Therefore, the goal is to look for the projected wire configuration that allows for the largest free area around the origin of the plane, which is the origin of the target object, where none of the nodes overlap with the projected area of the target object or end effector.

The wire model nodes are projected onto the plane of the target object using equation 36 and are evaluated on this plane. Recall that the target object plane is the plane normal to the grasp axis of the target object. The target object plane and the projections of points onto this plane were explained in sections 3.4.2 and 3.4.3. From the projected points, the first step was to quantify the

free/passable area that is allowed by the different wire configurations. To quantify this, the two end points and grasp point were queried from the wire configuration of interest, since they served as the boundary points of the wire configuration after a force was applied. A triangle can be made by connecting these three points, which captures the enclosed area made by the wire. However, this does not capture the full passable area, as shown in Figure 22, where the space under the triangle can also be considered as free/passable space.

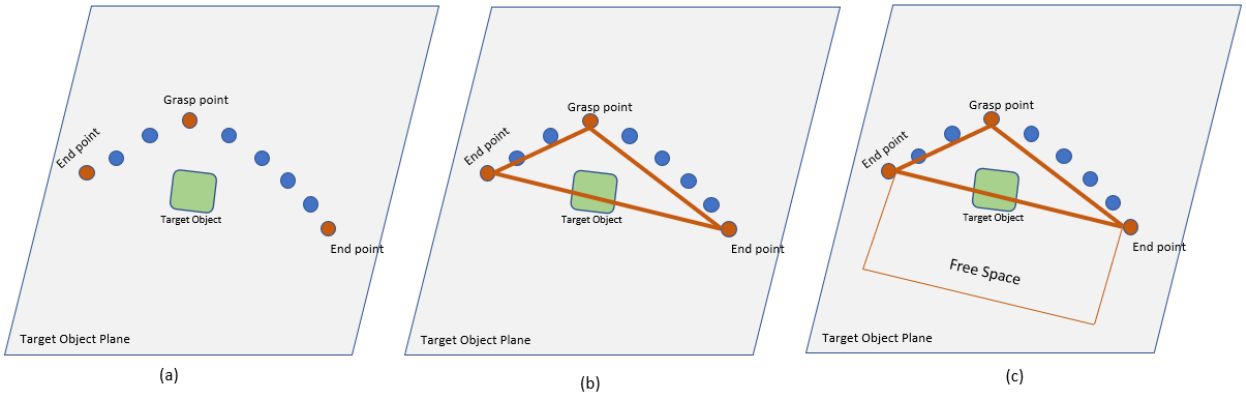


Figure 22: Projections onto target object frame

To ensure that the extra free space within the curve made by the wire is captured, the grasp point reflected across the axis made by the two end points is found in order to make a polygon. This can be visualized in Figure 23. This polygon is referred to as the *free space polygon*. This is a computationally fast and easy way to include the additional space within the curve made by the wire.

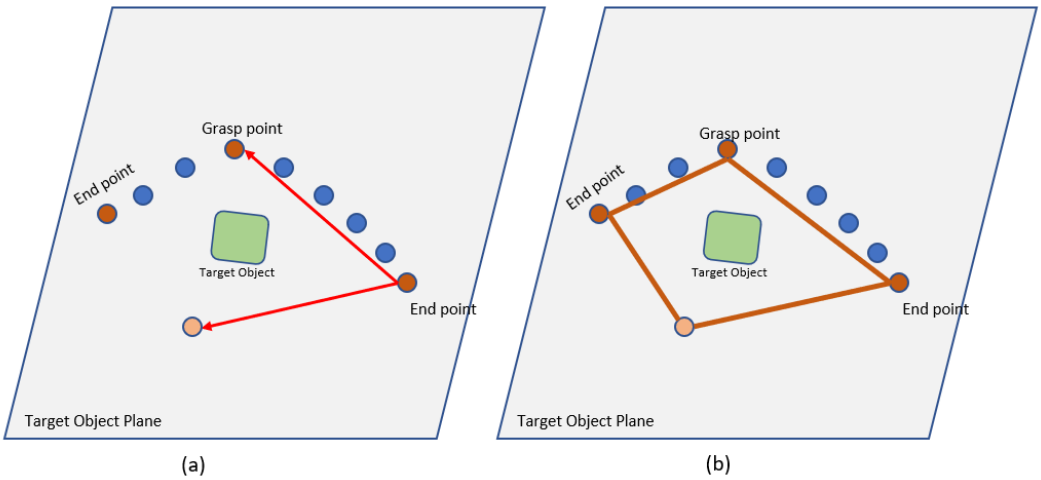


Figure 23: Creation of polygon

To generate the 2D free space polygon on the target object plane from the four points previously described, the python library *Shapely* was used. *Shapely* is a library designed to manipulate and evaluate geometric objects in a Cartesian plane. Similarly, using this library, 2D polygons are also made for the target object and the end effector projections on the Cartesian plane. These are referred to as the *target object polygon* and *end effector polygon*. Using a method from the *Shapely* library, the areas of the target object polygon, end effector polygon, and free space polygons are computed individually. With these various polygons, the goal is to find a free space polygon that encapsulates the target object and end effector polygons as shown in Figure 24. Such a configuration ensures that the target object and end effector can pass through collision free along the grasp axis of the target object.

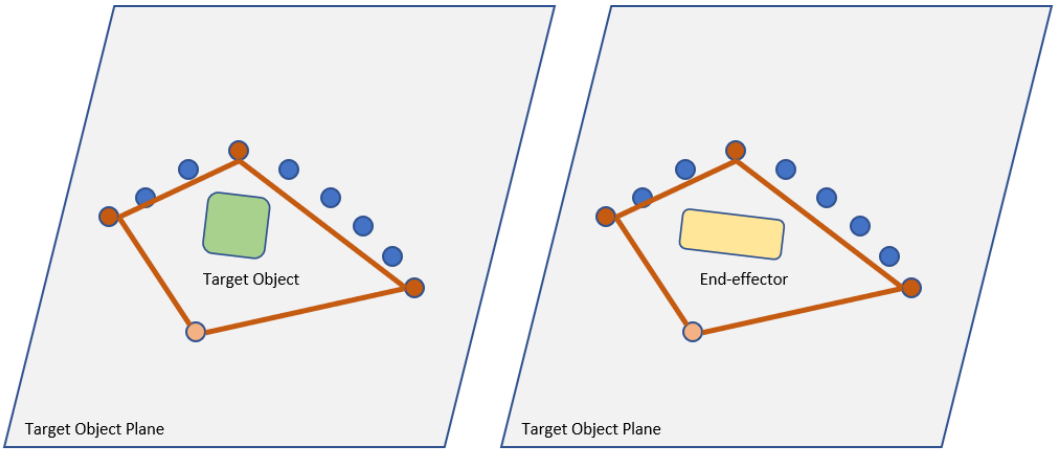


Figure 24: Comparing wire polygon to target object and end effector area

To determine the wire configuration that will allow this, the area of intersection between the free space polygon and the polygons made by the target object and end effector are computed individually. This is done for all remaining wire configurations in the set. If the area of intersection between the free space polygon and target object polygon is equivalent to the area of the target object polygon, then the free space polygon is said to fully encapsulate the target object polygon. The same is true for the robot end effector polygon. Therefore, the specific wire configuration that satisfies this requirement is identified as a configuration that allows for a collision free path to the target object. Figure 25 shows an example of a case where the free space polygon full encapsulates the polygon of the target object, and an example of when it does not.

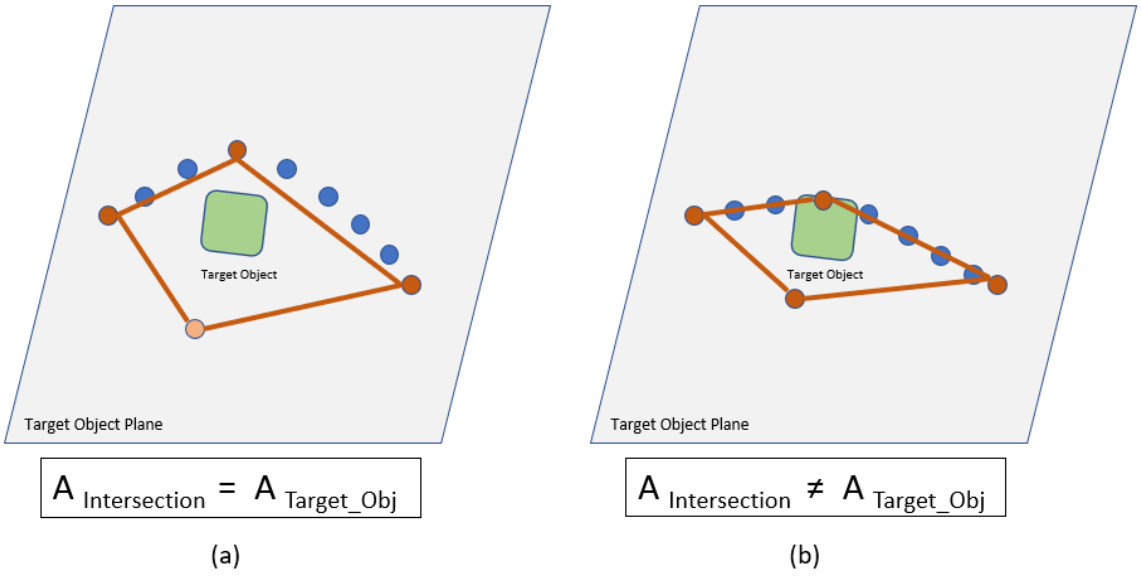


Figure 25: Finding area of intersection between polygons

After identifying a single wire configuration that does not collide with objects in the environment and allows for a path to the target object, the actions taken to achieve that wire configuration are retrieved and sent to the grasp planner and path planner for execution.

3.4.6 Task Allocation

With the pick and pull actions determined from the simulator, the next step is to determine which robot will manipulate the wire and which robot will grasp the object. A simple solution to this would be to dedicate one of the robots to only take actions on the wire, and the other robot to only grasp the target object. However, this approach is rather limiting and could lead to unwanted solutions where the robots need to cross each other. Therefore, the goal of the task allocation step is to assess the actions outputted by the simulator, and allocate given tasks to the two robots in order to avoid crossing of the arms in the workspace. To do this, the trajectory required to move the wire, which moves along the direction of the pull vector, is computed and evaluated with respect to the position of target object. If the end of the trajectory falls to the right of the target object in the world frame, then the task allocator assigns the right robot to grasp and move the wire. However,

if the end of the trajectory ends to the left of the grasp object, then the task allocator assigns the left robot to move the wire. Based on the result of this assignment, the task allocator will assign the other robot to grasp the target object. An executor then calls the necessary manipulation functions for the robots to execute. This execution step will be explained in greater detail in the next section.

3.5 Planning and Execution

3.5.1 Grasp Planning

With the grasp point and pull/push vector generated from the simulation model, fine motion plans are needed to carry out these high-level actions. The first step to execute these actions is to determine the robot end effector pose required to successfully grasp the wire. The 3D grasp point on the wire is given in world coordinates from the simulation model, however the challenging part is determining the orientation of the end effector to successfully grasp the wire at this point. For this, a custom grasp planner was developed specifically for grasping wires.

The ideal grasp of a wire with a 2-finger gripper is where the wire segment that is within the jaws of the gripper is parallel to the face of the gripper fingers. Figure 26 (a) shows the undesirable cases where the sides of the gripper fingers are perpendicular to the wire segment, whereas (b) shows the desirable case where they are parallel. Given this, the first objective of the grasp planner is to determine an orientation for the end effector, with respect to the world frame, such that the wire segment of interest is at the center of the gripper and parallel to the sides of the fingers.

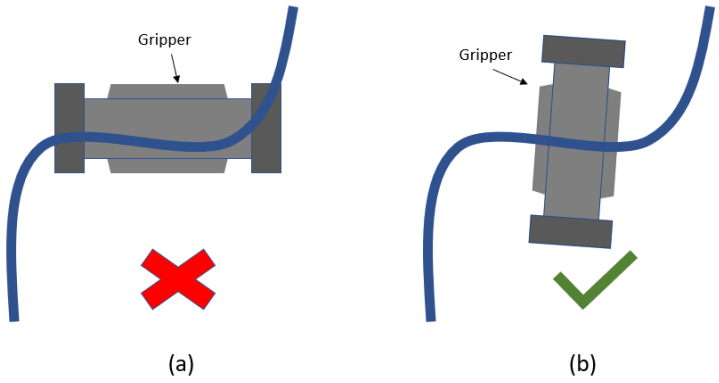


Figure 26: Ideal grasp orientation

Given the curve like nature of a wire, the grasp planner leverages Frenet-Serret frames, also known as Tangent, Normal, and Bi-normal (TNB) frames, to describe the orientation of a point on the curve. With the Bezier curve that is used to model the shape of the wire, a TNB frame can be constructed for any point on that curve. The process to compute a TNB frame for a given point on a Bezier curve was computed in section 3.4.3 using equations 37, 38, 39. Figure 27 displays the TNB frame for a given point on a Bezier curve, as well as in the world frame. Additionally, Figure 28 displays the frame used for the end effector, where the blue axis represents the z-axis, red axis represents the x-axis, and green axis represents the y-axis. By comparing the TNB frame in Figure 27 and the end effector frame in Figure 28, it can be observed that if the end effector frame was oriented such that the z-axis of the end effector was aligned with the Tangent vector, then wire segment at that point would be nearly parallel to the sides of the gripper as desired. Therefore, the first step of the grasp planner is to determine the end effector orientation that aligns the z-axis with the tangent vector.

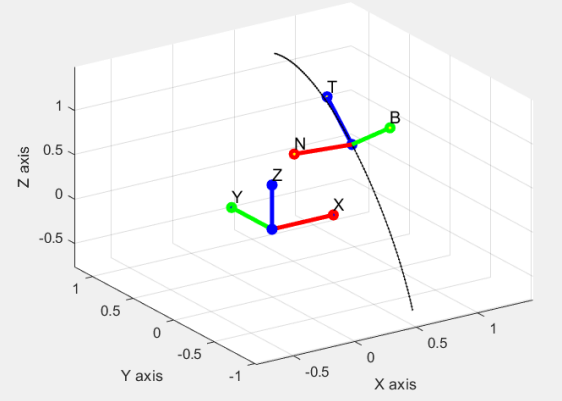


Figure 27: TNB frame

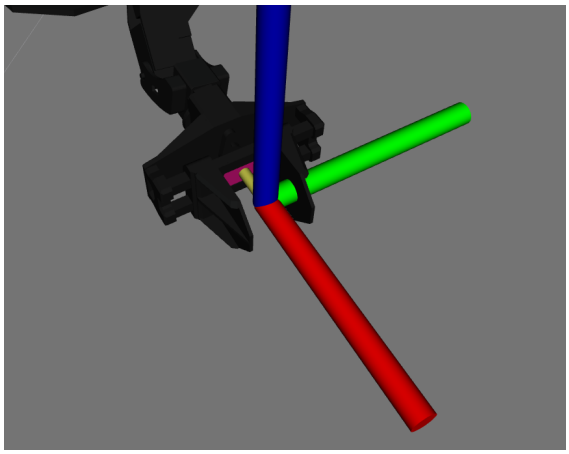


Figure 28: End effector frame

Given that the commanded end effector orientation is taken with respect to the world frame, the orientation of the TNB frame with respect to the world frame must first be determined. With the tangent, normal, and bi-normal vectors expressed in the world frame, a matrix can be made where the following vectors make up the columns of the matrix. Considering only the orientation of the TNB frame with respect to the world frame, equation 41 shows the relationship between the

two frames, where R is the rotation matrix. Filling in the terms in equation 41, equation 42 shows that the world frame is simply the identity matrix, I. Therefore, taking the inverse of the identity matrix on both sides, equation 43 shows that the rotation matrix between the two frames is simply the matrix made up of the tangent, normal, and bi-normal vectors.

$$\begin{bmatrix} x_{world} \\ y_{world} \\ z_{world} \end{bmatrix}^T * R = \begin{bmatrix} T \\ N \\ B \end{bmatrix}^T \tag{41}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * R = \begin{bmatrix} T_x & N_x & Bx \\ T_y & N_y & By \\ T_z & N_z & Bz \end{bmatrix} \tag{42}$$

$$R = \begin{bmatrix} T_x & N_x & Bx \\ T_y & N_y & By \\ T_z & N_z & Bz \end{bmatrix} \tag{43}$$

To determine the orientation of the end effector, the column vectors in the matrix in equation 43 are reorganized to achieve the desired end effector configuration. The x-axis of the end effector is mapped to the normal vector, the z-axis is mapped to the tangent vector, and the y-axis is mapped to the bi-normal vector. The outcome of this, shown in equation 44 is a rotation matrix that transforms the end effector to the desired configuration. Figure 29 showcases this transformation.

$$R_{ee} = \begin{bmatrix} N_x & B_x & Tx \\ N_y & B_y & Ty \\ N_z & B_z & Tz \end{bmatrix} \tag{44}$$

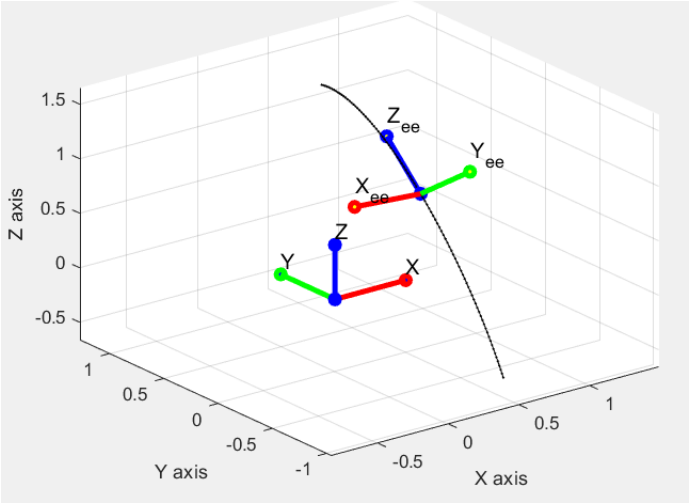


Figure 29: Grasp orientation

Although this rotation matrix ensures that the robot end effector orients itself to a position where the wire is parallel the gripper fingers, it does not consider the robot configuration required to achieve this grasp. The only criteria thus far for the grasp planner is to enforce that the z_{ee}-axis in the end effector frame is aligned to the tangent vector of the TNB frame. However, there are not yet any constraints on the rotation about the z_{ee}-axis, which could result in an orientation that is not physically possible. Moreover, the action after grasping the wire is to move the wire along the selected push vector. Therefore, the process for selecting an orientation for the end effector must consider how the end effector might interact with the wire when moving it along the push vector. To satisfy these two requirements, two constraints are imposed on the rotation of the end effector orientation about the z_{ee}-axis. The first is an equality constraint enforcing that the y_{ee}-axis should be parallel to the push vector, whereas the second constraint is an inequality constraint enforcing that the dot product between the x_{ee}-axis of the end effector frame and the x_{world}-axis of the world frame is greater or equal to zero. The constraints are shown below:

$$1 - |\vec{y_{ee}} \cdot \vec{v_{push}}| = 0 \tag{45}$$

$$\vec{x_{ee}} \cdot \vec{x_{world}} \geq 0 \tag{46}$$

Satisfying the first constraint, show in 45, requires an end effector orientation such that dot product of the y_{ee}-axis with the push vector computed from the simulator is equal to 1 or -1. In order to satisfy this constraint, the y_{ee}-axis must be parallel to the push vector, either in the same direction or directly in opposite directions. The reason for this can be explained by referring to Figure 30 of the end effector and end effector frame. When moving along the direction of the push vector, to avoid the wire slipping out of the gripper, the gripper sides should provide a normal surface for the wire to make contact with. Therefore, in order to provide a normal surface to make contact with the wire, the y_{ee}-axis must be near parallel to the direction of motion.

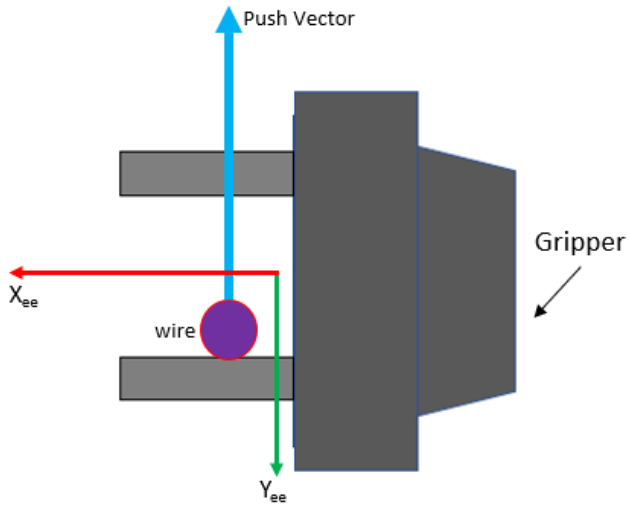


Figure 30: Aligning y-axis to pull vector

The second constraint, shown in equation 46, is enforced in order to avoid a robot configuration where the x_{ee}-axis is pointing back at the robot. This would not be a desired end effector configuration given that the robot would need to wrap behind the wire to achieve this end effector orientation, under the assumption that the robot is initially pointing in the direction of the x_{world}-axis. Therefore, this constraint forces the x_{ee}-axis of the end effector to point in the direction of the x_{world}-axis by ensuring that the dot product between the x_{ee}-axis and x_{world}-axis is positive.

To enforce these two constraints in the grasp planner, a grasp correction method is developed which takes the initial end effector orientation from equation 44, and applies small rotations of 0.5

degrees about the z_{ee}-axis until the constraints from equations 45 and 46 are met. Given the finite resolution of 0.5 degrees per rotation, it is unlikely that a end effector orientation will be found where the y_{ee}-axis will be perfectly parallel to the push vector. As a result, a small *threshold* is defined so that the implementation of the first constraint is now $1 - |\vec{y_{ee}} \cdot \vec{v_{push}}| \leq Threshold$. To get the best result, the threshold is set to a small value, 0.01 in this case, such that an end effector orientation, where y_{ee}-axis very near parallel can be found. The grasp correction method applies these small rotations and checks the constraints in a loop that terminates when the two conditions are met, or terminates after $\frac{360}{resolution}$ iterations which indicates that the end effector frame was rotated 360 degrees without satisfying the constraints. In this case, given the resolution, the grasp correction failed to find a y_{ee}-axis that was parallel enough to satisfy the defined threshold. To avoid failure at this point, the main loop that applies the small rotations and checks the constraints is wrapped within another loop that increases the threshold value until an end effector orientation that satisfies the two constraints can be found. The pseudo code for the grasp correction method is shown below, along with Figures 31 and 32 which show the end effector frame before and after applying the grasp correction method. In Figure 32 we can see that the y_{ee}-axis is nearly parallel to the push vector while the x_{ee}-axis points in the $+x_{world}$-axis.

Algorithm 3 Grasp Correction

```
while Condition = True do
    for i = 1, 2, ..., 360/resolution do
        Apply rotation of 0.5 degrees about the z_ee - axis
        if Both constraints are satisfied then
            Store end effector orientation
            While loop Condition = False
            Break for loop
        end if
    end for
Tolerance = Tolerance + 0.01
end while
```

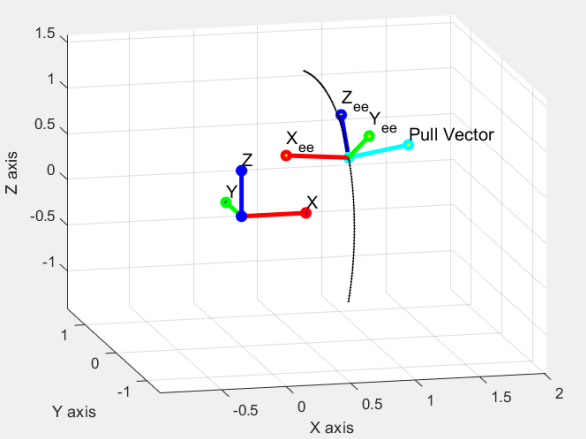


Figure 31: Computed orientation from TNB frame

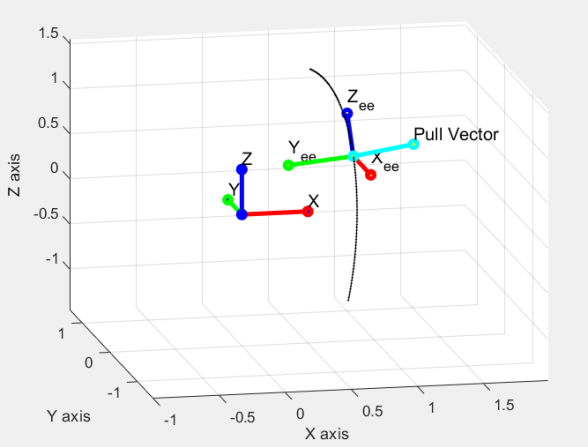


Figure 32: Correction orientation

3.5.2 Motion Planning and Execution

To execute the high-level pick and pull actions output from the simulator, the Moveit motion planning framework is used which provides motion planners, collision checking/avoidance, and trajectory execution capabilities. The controller of choice for this work is a position-based controller where joint positions are computed to move the end effector of the arms to desired 6DOF poses defined from the motion planner. It is important to note that no force controller is used in this work, only purely position control. The motion planner used in this work is the Rapidly-exploring random trees star (RRT*) algorithm paired with the Flexible Collision Library (FCL) for collision checking.

The Moveit API is used to create ROS services for grasping the wire, moving the wire, and grasping the target object. An executor is designed to move the system through different states and executes these primitive ROS services along with ROS services in the vision pipeline based on the given state. A high level explanation of these services is provided below.

ROS SERVICE: Grasp Wire	
Input	6DOF end effector pose from the wire grasp planner
Function	Moves robot to a pre-grasp configuration, offset from the desired grasp pose, then moves along a Cartesian path to the grasp pose.

ROS SERVICE: Move Wire	
Input	3D pull vector outputted from the simulator
Function	Moves end effector along a Cartesian path in the direction of the pull/push vector

ROS SERVICE: Grasp Target Object	
Input	6DOF pose of the target object
Function	Samples through a repository of pre-defined grasp poses and plans paths to the grasp pose until a collision free path to the desired object is found. The robot then moves to a pre-grasp configuration, moves to a grasp configuration, then moves to a post grasp configuration.

A diagram of the overall system is shown in Figure 33. The system first begins by obtaining a raw point cloud of the environment which is passed through the segmentation node to segment the wire from the environment. The point cloud of the wire is then passed to the state estimation node, which constructs a model of the wire from a point cloud. The model is then passed to the simulator node which generates a dynamic model of the wire and environment scene, and simulates different pick and pull actions until a pair of actions satisfies defined constraints. These actions are passed to the executor node which determines which robot should perform which task, and calls the robot services in sequence to manipulate the wire and grasp the target object.

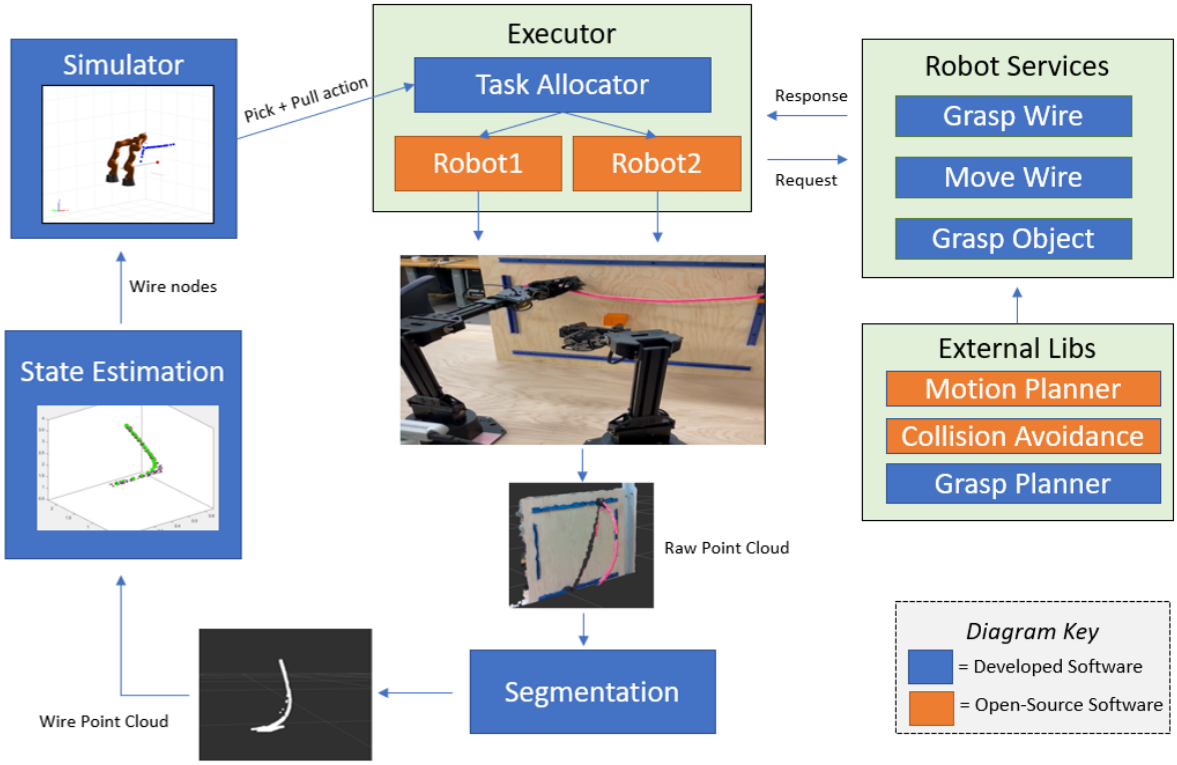


Figure 33: Overall Software Architecture

4 Results

The complete system was tested on hardware with two Trossen ViperX 300s robots and an Intel RealSense d435i camera. The experimental mock-up was previously described in section 3.1.1. In this experiment, the target object was a cube for simplicity. It was assumed that the robotic arms had the capability to position themselves in front of the wire/panel of interest, and therefore only a configuration where the robots were directly facing the panel was only considered in this experiment. This experiment evaluated the accuracy of the vision pipeline in constructing a 3D models of the wire from a 2D image, as well as tested the performance of the complete system in moving an obstructing wire to clear a path to a target object. Section 4.1.1 covers the results from the vision pipeline and section 4.2.1 covers the overall performance and repeatability of the system in moving the obstructing wire.

4.1 Vision

4.1.1 Experiment Overview

The goal of the vision pipeline was to sense and estimate the state of the wire in the environment in order to create a 3D representation of that wire with a set of nodes. In this experiment, the key metrics of interest to evaluate the vision system were robustness and accuracy. To evaluate the accuracy of the vision system, the 3D wire model output by the vision system was compared to the ground truth position of the wire, and errors in position were recorded. In order to get the ground truth position of the wire, 10 Apriltag fiducial markers [26] were attached and spaced evenly along the body of the wire, allowing for the 3D pose of each tag to be recorded with a detection algorithm. Figure 34 shows a visual of this process. A Bezier curve was then fit to these 10 points and compared to the Bezier curve output by the vision system. The Euclidean distance between points on the two curves were computed and averaged to form a single average position error value along with the variance which quantified the accuracy in the vision system. This was performed for the position of the wire in 3D as well as in 2D. To test the vision systems robustness, this experiment was performed on three different wire thicknesses: 0.25in, 0.375in, and 0.5in. Sections 4.1.2, 4.1.3, and 4.1.4 show the results for the three wire thicknesses.

Figure 34: ApriTags used to obtain ground truth position of wire

4.1.2 Vision Pipeline Results: 0.5in Wire Thickness

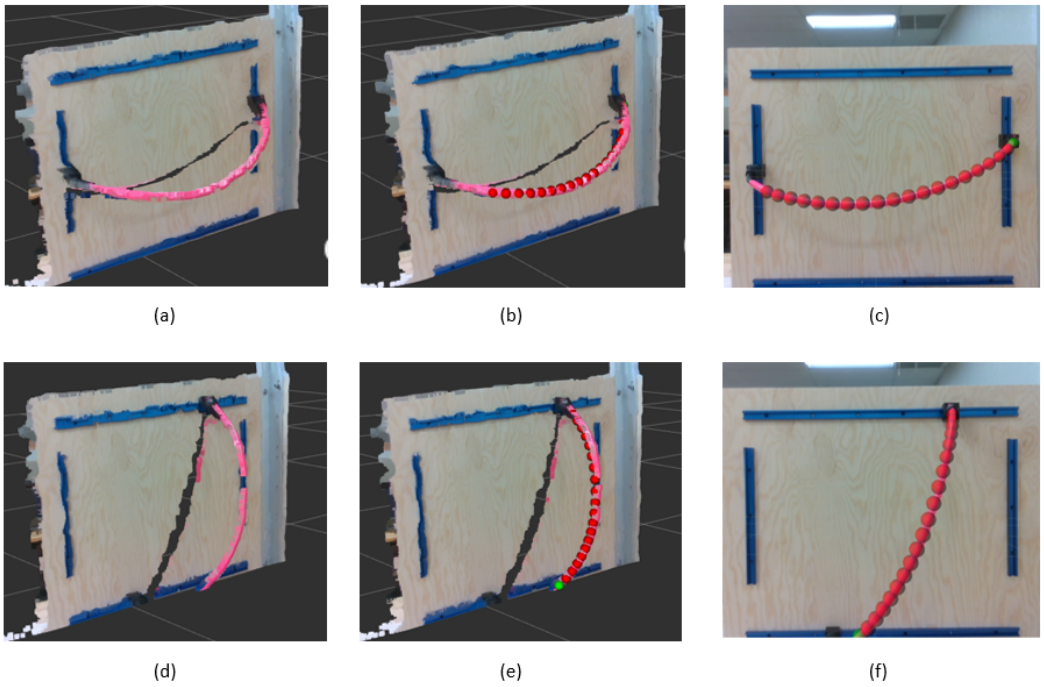


Figure 35: **Results for Case 1: 0.5in wire** (a) Horizontal: raw point cloud (b) Horizontal: 3D estimation (c) Horizontal: 2D estimation (d) Vertical: raw point cloud (e) Vertical: 3D estimation (f) Vertical: 2D estimation

Position Error (in)	**Mean Error**	**Variance**
2D	0.141	1.57e-4
3D	0.381	1.07e-3

Table 1: 0.5in Horizontal Wire

Position Error (in)	**Mean Error**	**Variance**
2D	0.078	3.14e-5
3D	0.173	2.1e-4

Table 2: 0.5in Vertical Wire

4.1.3 Vision Pipeline Results: 0.375in Wire Thickness

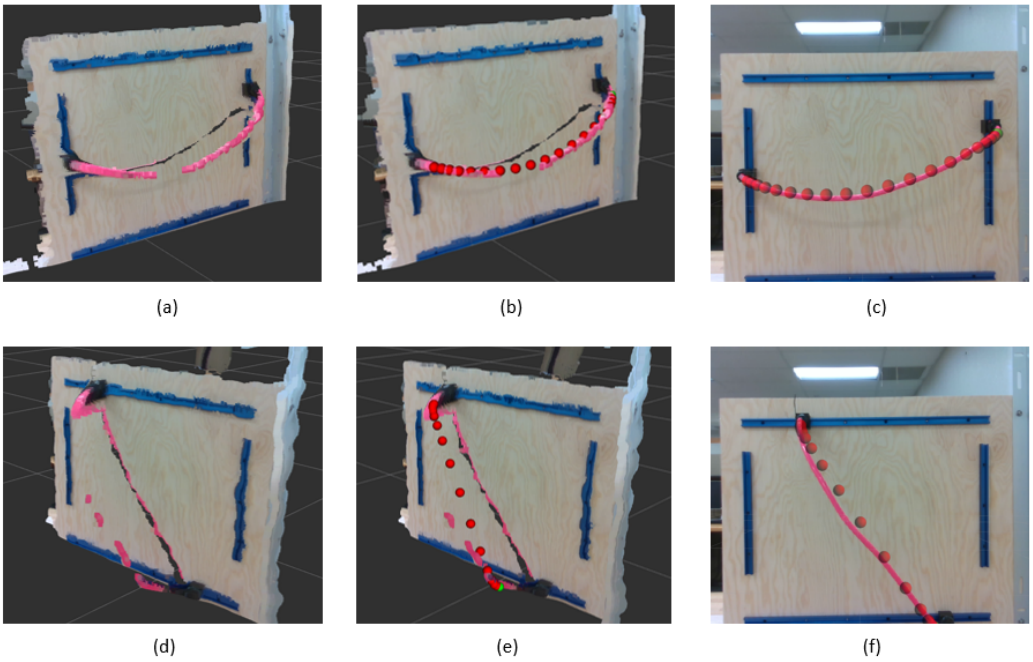


Figure 36: **Results for Case 1: 0.375in wire** (a) Horizontal: raw point cloud (b) Horizontal: 3D estimation (c) Horizontal: 2D estimation (d) Vertical: raw point cloud (e) Vertical: 3D estimation (f) Vertical: 2D estimation

Position Error (in)	**Mean Error**	**Variance**
2D	0.145	1.37e-4
3D	0.342	5.9e-3

Table 3: 0.375in Horizontal Wire

Position Error (in)	**Mean Error**	**Variance**
2D	0.511	2.1e-3
3D	1.63	7.87e-4

Table 4: 0.375in Vertical Wire

4.1.4 Vision Pipeline Results: 0.25in Wire Thickness

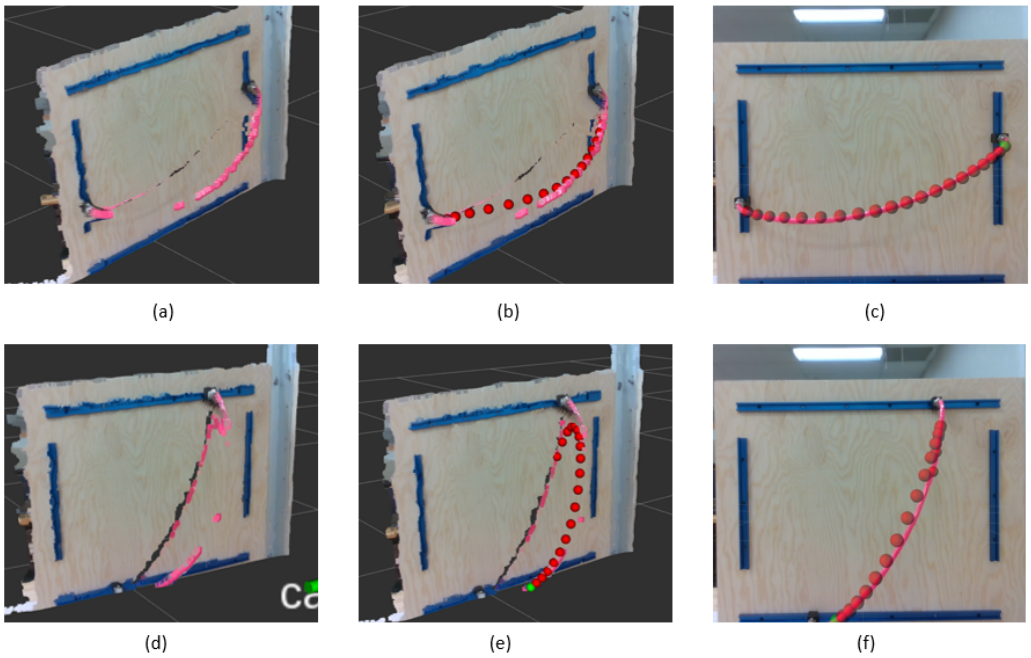


Figure 37: **Results for Case 1: 0.25in wire** (a) Horizontal: raw point cloud (b) Horizontal: 3D estimation (c) Horizontal: 2D estimation (d) Vertical: raw point cloud (e) Vertical: 3D estimation (f) Vertical: 2D estimation

Position Error (in)	**Mean Error**	**Variance**
2D	0.334	4.2e-4
3D	0.55	1.05e-3

Table 5: 0.25in Horizontal Wire

Position Error (in)	**Mean Error**	**Variance**
2D	0.669	2.4e-2
3D	1.18	4.3e-2

Table 6: 0.25in Vertical Wire

4.2 Planning and Control

4.2.1 Experiment Overview

In this experiment, the entire system's performance in sensing an unknown scene and moving an obstructing wire to create a clear path to a target object was evaluated. The metrics of interest are the system's success rate and robustness in sensing and manipulating the wire to create a path to the target object. To measure the success rate, the system was run 30 times, where each trial contained a different wire configuration. The system was evaluated on a 'success' or 'fail' basis, where a 'success' was awarded if the robot successfully moved the wire to clear a path, and a 'fail' was given if the robot could not clear a path to the target object. In the case where the robot failed, the fail was classified as either failing due to 'vision' or failing due to 'planning'. Failing due to 'vision' was determined by visually observing the estimated wire modeling outputted by the vision pipeline. If the estimation of the wire position was poor and significantly differed from the ground truth, and the robot failed to clear a path, then this was considered a failure due to vision. However, if the estimation of wire position outputted by the vision pipeline aligned with the ground truth and the system failed to clear a path, then this was considered to be a failure due to planning. To evaluate the robustness of the system, this experiment was performed for three different wire thicknesses: 0.25in, 0.375in, and 0.5in.

4.2.2 Planning and Control Results

Figure 38 below shows the steps of a successful clearing of the wire to create a path to the orange target object. From the first image, the initial state of the wire and target orange object can be observed. In this case, the grasp axis of the target object points outward and normal to the panel. Therefore, it is apparent that the wire initially obstructed the path to the orange object and the robot was initially unable to remove the object along that grasp axis. The state of the environment was passed to the simulator, along with the state of the wire obtained through vision, and the simulator returned a grasp point on the wire and pull/push vector. Based on the initial state of the

environment and simulated actions to be taken, the task allocator assigned the left robot to grasp the wire and the right robot to grasp the target object. The grasp planner then planned a grasp for the wire, which can be seen in the first image. From the middle image, the robot then moved the wire along a trajectory in the direction of the determined push/pull vector. In the last image, once the area was cleared, the second arm then computed a collision free motion plan to reach the target object.

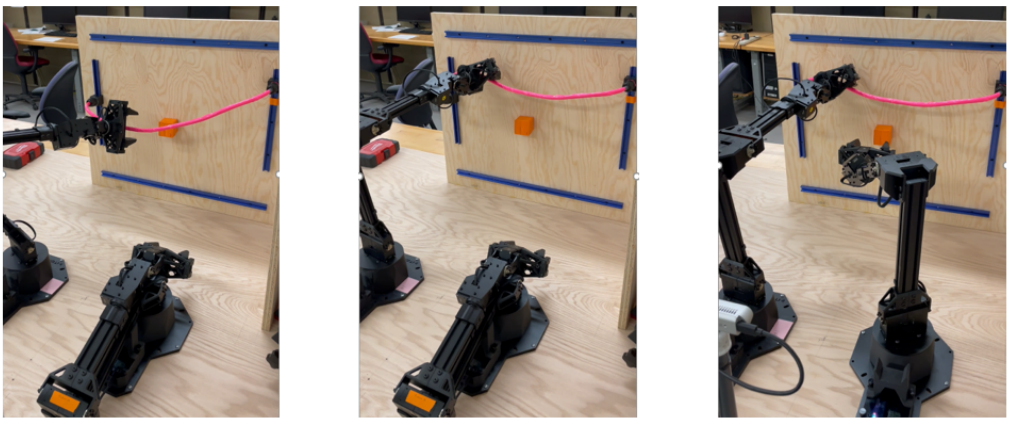


Figure 38: **Demo**

Figure 39 shows a bar graph of successes versus failures for the three wire thicknesses out of 30 trials for each wire thickness. Table 7 displays the success rate for each wire thickness.

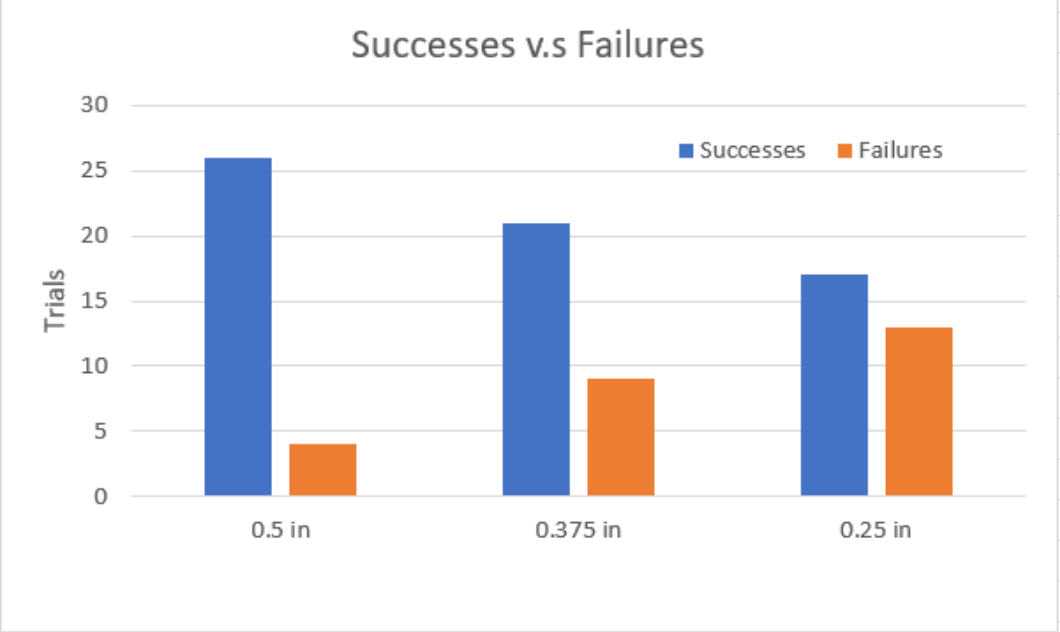


Figure 39: **Successes vs. Failures**

Wire Thickness	Success Rate
Wire1	86.667 %
Wire2	70.000 %
Wire3	56.667 %

Table 7: Percentage of success for the three wire thicknesses

Figure 40 shows a bar graph displaying the makeup of failures due to vision, versus failures due to planning, for the three wire thicknesses. The bar height indicates the total number of failures for each wire thickness. The blue bars represent failures due to planning and the orange bars represents failures due to vision for the three wire thicknesses.

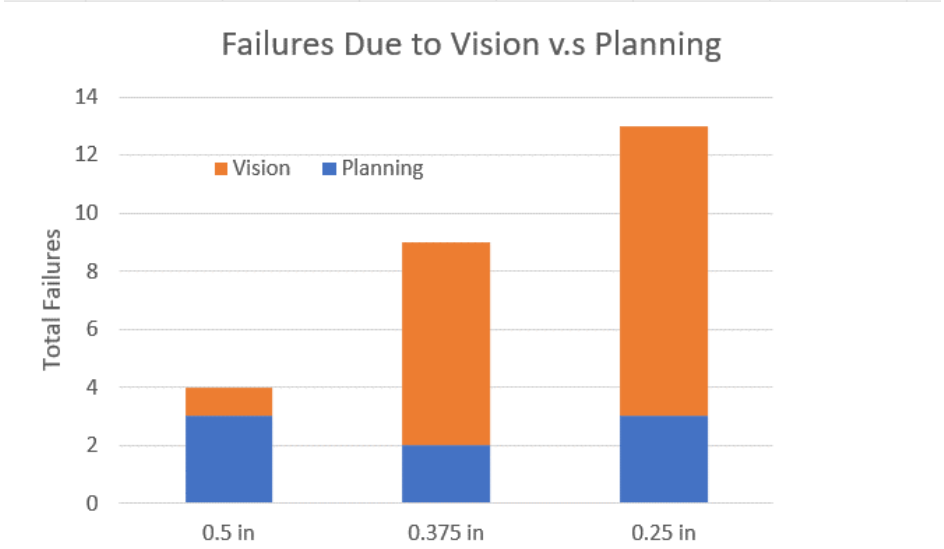


Figure 40: **Failures due to Vision vs. Planning**

4.3 Discussion

4.3.1 Vision

In analyzing the results for the vision system in section 4.1.1, the vision system showed good performance for wire thicknesses 0.5in and 0.375in, and showed difficulty in sensing the 0.25in thick wire. In all the results for all three wire thicknesses, what is evident is that if the camera is able to capture an accurate representation of the wire in the point cloud, then the vision pipeline is able to estimate the position of the wire with minimal errors in position. However, since RGB-D cameras in general are sensitive to influences such as light intensities, reflective characteristics, and viewing angles, the point cloud created from the depth image will inevitably contain outliers that do not belong on the surface of the object and will contain significant errors in the sensed depth. From Figure 37 (d), it is evident that there is a significant error associated with the initial point cloud of the wire. Portions of the pink wire are detected to be on the surface of the panel, which presents an issue for the vision system if it is unable to reject those outliers. These errors in depth are a result of the limitations of the hardware, and not as a result of failure in segmentation. Since the segmentation occurred in a 2D RGB image and was used as a mask to segment a depth image,

everything in the depth image besides the pixels associated with the wire were rejected. Therefore, the errors in the depth values for the pixels associated with the wire object went undetected and were passed on to be converted into a point cloud. An alternative approach to segmentation of a point cloud would be to perform some form of clustering or to apply deep learning techniques directly on the point cloud. However, regardless of the method of segmentation of point clouds, a point cloud similar to that in Figure 37 (d) would present issues for any method, given the lack 3D points on the wire captured by the sensor.

An example where the vision system showed robustness is in Figure 36, which showcases the results for the 0.375in wire thickness case. In Figure 36 (d), it can be observed that the full point cloud of the wire was not captured due to reasons previously discussed. Though not entirely accurate, the vision system was still able to output a reasonable representation of the wire as shown in Figure 36 (e). The reason for this is due to the curve fitting step, where a curve is able to connect the points output from an initial clustering step on the segmented point cloud. So for point clouds that partially capture the body of the wire as in Figure 36 (d), the clustering step generates points that make up the partially captured body of the wire, which is enough to fit a curve to estimate the end to end shape of the wire. This allows for gaps in the shape of the wire to be filled in. Of course, the better the initial point cloud of the wire, the better the curve fit which is why the estimation of the wire shape in Figure 36 (e) is slightly off. Nonetheless, it still shows robustness in modeling the wire shape from a poor initial point cloud of the wire.

With the main issue being the generation of an accurate point cloud of the wire, an approach to improve the point cloud would be to capture multiple images of the scene from different camera angles and fuse the resulting point clouds together. By doing so, segments on the wire that were not captured in one camera angle could be captured in the point cloud as a result of a different camera angle. In theory, this will lead to a more complete representation of the wire in a point cloud and will result in more consistent and repeatable results, similar to those shown in this work where the raw point cloud captured the majority of the wire.

4.3.2 Planning and Control

In evaluating the results for the success rate of the system, it can be observed that the performance of the system degrades as the thickness of the wire decreases. In evaluating Figure 40, we can see that the failures due to vision increases significantly as the thickness of the wire decreases, which explains the decrease in performance as the wire thickness decreases. Sensing thin wires appears to be the limitation in this system and remains an open research problem.

Looking at the 0.5in wire thickness case, where the vision system could capture the state of the wire with low uncertainty, the system showed an 86.667% success rate out of 30 trials of random wire configurations. This case captures a good representation of the performance of the task and motion planning, given that only one failure was a result of the vision system. In the other two cases it is difficult to assess the true performance of the planning portion of the system, since the majority of the failures were due to vision, which feeds into the planning pipeline.

In some cases, the robot generates a plan which pushes the wire to an ideal location; however it might push too far and damage the wire or the wire may slip out of the jaws of the robot gripper. In these cases, this was recorded as a failure due to planning, since the planner failed to predict the limits of the wire. However, the limits at which we can manipulate the wire can arguably be handled at a reactive level as opposed to trying to determine those limits in the planning layer. Equipping the robot with a force-torque sensor can allow for force feedback which can assist the robot in avoiding failures such as those previously described. Considering this, along with the failure due to vision, the total performance of the task and motion planning in the system shows promising results.

5 Conclusion and Future Work

5.1 Conclusion

The research presented herein presents a dual arm robotic system that can clear a wire that is obstructing the path to a target object. The system is capable of segmenting a wire from a point cloud, by performing segmentation of a wire in a 2D RGB image and converting that to 3D space. The wire was colored pink and segmented from the background in RGB images based on HSV color filtering. This method can easily be replaced with a more robust and general method for segmenting a wire, or multiple wires, from the environment in a RGB image. Segmentation of the wire in the RGB 2D image, then converting that into a point cloud, was chosen for this work due to the errors in the raw point cloud of the wire. Though not tested, it can be observed that segmentation of the wire directly on the point cloud would be difficult when errors in the raw point cloud exist, given that the wire is not rich in features. From the results presented in section 4.1.1, it can be observed that the segmentation method chose for this work performed well in rejecting outliers and noise in the raw point cloud.

The novelty in this work is the capability to construct a model of the wire, purely from vision, with no prior knowledge of the wire length. The system demonstrated an ability to convert a point cloud of the wire to a chain of 20 nodes which characterize the wire. The system can reject outliers presented in the data and generate a chain of nodes which forms a smooth curve to better represent the characteristics of a wire. In addition, from this wire model, this work demonstrated the ability to create a mass spring dynamics model of the wire. This dynamical model of the wire is used in a light-weight physics-based simulation which is used to simulate different actions on the wire. The simulation environment was used to determine the best pair of pick and pull actions to apply to the wire, which resulted in clearing a path to a target object. Based on the outputs from the simulator and environment state, a task allocator selects which arm will move the wire and which arm will grasp the target object. The job of the task allocator is to prevent the crossing of the arms when executing the tasks, which is a crucial part of the system. Crossing of the arms would further

complicate the motion planning for the arms and further restrict the path to the target object.

To grasp the wire, a grasp planner was developed which solved for an end effector pose that wrapped the gripper fingers around the wire at a defined point on the wire. The grasp planner also ensured that a valid grasp was returned by enforcing constraints on the grasp orientation. To execute the robot actions, pre-programmed primitive actions were developed, using ROS and moveit, for wire grasping and moving, as well as grasping of the target object. These actions could be called and applied to either robot, therefore allowing for either robot to grasp and move the wire.

Overall, the system showed notable success in sensing, planning, and clearing an obstructing wire in a workspace. The system performed better on wires with a thickness of 0.375in and above, and showed reasonable results for wires with thicknesses less than that. For a wire thickness of 0.5in, the system demonstrated a 86.7% success rate in clearing the wire. The limitation in performance was due to vision, specifically due to the fact that only one camera view was used to obtain a point cloud of the wire. With a good initial raw point cloud, that captures most of the body of the wire, the planning system demonstrates an ability to find a pair of actions to clear the obstructing wire in order to reach a target object.

5.2 Future Work

A significant limiting factor in this research was the performance of the vision system to estimate the state of thin wires. This was due to limitations in the physical hardware, as well as the fact that the segmentation was only applied on a single RGB image at a single camera angle. An immediate area of improvement for this system is to introduce a point cloud segmentation scheme that utilizes multiple camera angles. Having multiple camera angles allows for a better representation of the scene in a point cloud, which will allow for feasible outlier detection and better state estimation by comparing multiple point clouds of the environment.

An addition to this project that was explored, but did not quite make it into this paper, was utilizing Deep Learning to perform segmentation of a wire in a RGB image. This work exper-

imented with applying a DeepLabv3 [27] model trained on an existing labeled data set [10] to perform semantic segmentation of electrical wires in an RGB image. This model showed very promising results, however issues arose when integrating the model with the existing robot software running in ROS noetic given the jump to Python 3.7. The Deeplabv3 model contained a ROS wrapper centered around Python 2.7, which complicated the integration. The reason for choosing the DeepLabv3 model was influenced by the labeled training data provided by [10] which contained a label suitable for the Deeplabv3 model. Creating a new data set for wire segmentation was out of scope for this work. Therefore, the integration of the Deeplabv3 model in this work was labeled as experimental. The vision pipeline in this work is designed to be modular enough so that binary mask outputs from any segmentation method, that does not have conflicting dependencies, can be plugged in. Therefore, a finishing next step to extending this work would be to complete the integration of Deep Learning in the vision pipeline.

To improve the wire model, aspects of the wire such as the curvature can be further explored to draw conclusions on the stiffness of the wire along its body. Areas with higher curvature will be more elongated and stiffer than areas with lower curvature. This can be important in the wire simulation in order to increase the accuracy of the simulation by assigning stiffnesses along the body depending on the shape of the wire. In addition, areas on the wire with high curvature can also signify potential areas with high ranges of motion. This can be used to determine potential grasp points on the wire.

A big area of improvement is in the robot controllers used for manipulation. Currently, only position controllers are used, which move the arm to defined way points determined by the planner based on the wire simulation. This is problematic, since the robot has no force feedback and can either damage itself or the wire. Therefore, deploying a hybrid force-position controller or an impedance controller would be ideal for this use case, which can be used to adjust the robot's trajectory during manipulation based on the forces applied by the wire. In addition, having force feedback can allow for system identification of the wire, which can be determine the stiffness of the wire accurately, as opposed to estimating stiffness from vision.

References

[1] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[2] K. Alsabti, S. Ranka, and V. Singh, "An efficient k-means clustering algorithm," 1997.

[3] B. Yuan, H. Du, H. Wang, and W. Xiong, "The simulation of cable harness based on mass-spring model," in *MATEC Web of Conferences*, vol. 31. EDP Sciences, 2015, p. 10002.

[4] T. Tang, Y. Fan, H.-C. Lin, and M. Tomizuka, "State estimation for deformable objects by point registration and dynamic simulation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2427–2433.

[5] T. Tang, C. Wang, and M. Tomizuka, "A framework for manipulating deformable linear objects by coherent point drift," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3426–3433, 2018.

[6] M. G. Bualat, T. Smith, E. E. Smith, T. Fong, and D. Wheeler, "Astrobee: A new tool for iss operations," in *2018 SpaceOps Conference*, 2018, p. 2517.

[7] J. F. Russell and D. M. Klaus, "Maintenance, reliability and policies for orbital space station life support systems," *Reliability Engineering & System Safety*, vol. 92, no. 6, pp. 808–820, 2007.

[8] W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto, "Learning predictive representations for deformable objects using contrastive estimation," *arXiv preprint arXiv:2003.05436*, 2020.

[9] M. Yan, Y. Zhu, N. Jin, and J. Bohg, "Self-supervised learning of state estimation for manipulating deformable linear objects," *IEEE robotics and automation letters*, vol. 5, no. 2, pp. 2372–2379, 2020.

[10] R. Zanella, A. Caporali, K. Tadaka, D. De Gregorio, and G. Palli, "Auto-generated wires dataset for semantic segmentation with domain-independence," in *2021 International Conference on Computer, Control and Robotics (ICCCR)*. IEEE, 2021, pp. 292–298.

[11] Y. Li, "Object detection and instance segmentation of cables," 2019.

[12] L. Keselman, J. Iselin Woodfill, A. Grunnet-Jepsen, and A. Bhowmik, "Intel realsense stereoscopic depth cameras," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2017, pp. 1–10.

[13] C. Jia, T. Yang, C. Wang, B. Fan, and F. He, "A new fast filtering algorithm for a 3d point cloud based on rgb-d information," *PloS one*, vol. 14, no. 8, p. e0220253, 2019.

[14] B.-Q. Shi, J. Liang, and Q. Liu, "Adaptive simplification of point cloud using k-means clustering," *Computer-Aided Design*, vol. 43, no. 8, pp. 910–922, 2011.

[15] S. Baydas and B. Karakas, "Defining a curve as a bezier curve," *Journal of Taibah University for Science*, vol. 13, no. 1, pp. 522–528, 2019.

[16] E. Erkan and S. Yüce, "Serret-frenet frame and curvatures of bézier curves," *Mathematics*, vol. 6, no. 12, 2018.

[17] X. Provot *et al.*, "Deformation constraints in a mass-spring model to describe rigid cloth behaviour," in *Graphics interface*. Canadian Information Processing Society, 1995, pp. 147–147.

[18] A. Loock, E. Schömer, and I. Stadtwald, "A virtual environment for interactive assembly simulation: From rigid bodies to deformable cables," in *5th World Multiconference on Systemics, Cybernetics and Informatics (SCI'01)*, vol. 3. Citeseer, 2001, pp. 325–332.

[19] N. Lv, J. Liu, H. Xia, J. Ma, and X. Yang, "A review of techniques for modeling flexible cables," *Computer-Aided Design*, vol. 122, p. 102826, 2020.

[20] A. Sledd and C. Mueller, "Express rack overview," in *37th Aerospace Sciences Meeting and Exhibit*, 1999, p. 313.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[22] D. J. Bora, "A novel approach for color image edge detection using multidirectional sobel filter on hsv color space," *Int. J. Comput. Sci. Eng*, vol. 5, no. 2, pp. 154–159, 2017.

[23] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of robot 3d path planning algorithms," *Journal of Control Science and Engineering*, vol. 2016, 2016.

[24] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[25] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 3859–3866.

[26] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 4193–4198.

[27] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.