Norges miljø- og biovitenskapelige universitet

Master's Thesis 2024 30 ECTS
Faculty of Science and Technology

Potential Field-Based Path Planning for Enhanced Sensor Coverage in Robotic Salmon Inspection

Daniel Glemminge
Applied Robotics

Preface

This Master's Thesis marks the completio of my education in Applied Robotics at the Norwegian University of Life Sciences (NMBU). I would first like to thank my knowledgeable supervisors, Dr.Antonio Candea Leite, Michael Angelo Amith Fenelon, and Dr.Jens Petter Wold. They have given me the opportunity to be a part of the DigiFoods research project, and provided support and feedback throughout the many hours of work put into this thesis.

Additionally, the work put into the thesis has led to the writing of a separate research paper, which will be submitted to an automation conference in the near future. In this regard, I would like to give a special thanks to Michael for the many hours spent together in the robotics lab at NMBU, and to Antonio for the thorough feedback. I wish you both the very best of luck in the future.

I would also like to thank my family for always being there for me, and pushing me to reach my fullest potential. They have always believed in me, and I can not express enough gratitude for this. Lastly I want to thank my friends for many social activities and support during these last five years, and I look forward to more of this in the future.

This work is partially supported by the Research Council of Norway (NFR) through the project SFI Digital Food Quality (DigiFoods), Project Number 309259.

Wishing you an engaging journey through these pages

Daniel Glemminge

Ås, May 15th, 2024

Abstract

Salmon constitutes a large part of the fish industry, and the increasing number of salmon fillets with melanin spots leads to significant economic losses due to reduced quality perception by the customers. Studies show that Omega-3 Fatty acid content is correlated with a more healthy-looking fillet, and it has been proven that Raman spectroscopy is feasible for estimating this content. Further research shows that a robotic solution is very promising to overcome the tedious and error-prone labor of manual scanning in a production line. If these melanin spots are not avoided during the spectroscopic measurements, the resulting spectra can be misleading and imprecise.

This motivates the development of an enhanced sensor coverage solution, which led to the research question: “How can machine vision, AI-driven models, image processing, and path planning be used to generate a scan profile and instruct a robot arm to perform an effective scanning of salmon fillets, ensuring the avoidance of melanin spots?”

To reach this goal, a robotic pipeline was built, which utilizes Mask R-CNN to segment the belly and pseudo-melanin spots on the salmon fillet, followed by image processing and a step-wise path planning method inspired by artificial potential fields (APFs) method. The APF-like path planner is used to command a 6-DoF robot arm in a Gazebo-simulated environment using the motion planning framework “Moveit”.

The results demonstrate a planning algorithm that is able to generate complex paths and adapt to multiple melanin spots and orientations of the fillet. The AI-driven image segmentation needs some improvements, and the use of pseudo-melanin spots due to the unavailable dataset makes it difficult to assess its full performance. The robotic scanning is able to accurately follow the planned path with an average execution time of 3.365 seconds, corresponding to an average TCP speed of 116.3 $mm\ s^{-1}$, which is within the required exposure time for a satisfactory Raman scanning.

Future work in this field of research includes using a relevant dataset for more precise training, testing and validation of the AI-driven models and the APF-like path planner. Furthermore, when testing in a laboratory environment with a Raman spectroscopy sensor and a moving conveyor belt, the robotic system can be further tuned and optimized, making it possible to implement the solution in a relevant environment.

Sammendrag

Laks utgjør en stor andel av fiskeindustrien, og den økende forekomsten av laksefilleter med melaninflekker forårsaker store økonomiske tap grunnet nedsatt kvalitetsoppfatning hos kundene. Studier viser at Omega-3 fettsyreinnhold er korrelert med en fillet som ser sunnere ut, og det har blitt bevist at det er mulig å bruke Raman spektroskopi for å estimere dette innholdet. Videre studier viser at en robot-løsning ser veldig lovende ut for å løse utfordringen med det langtekkelige og feiltilbøyelige arbeidet med denne manuelle skanningen ved en produksjonslinje. Hvis disse melaninflekkene ikke blir unngått under spektroskopien, kan de resulterende spektrene være villedende og av mindre verdi.

Dette motiverer utviklingen av en forbedret sensor-dekkende løsning, som ledet til det følgende forskningsspørsmålet: "Hvordan kan maskinsyn, KI-drevne modeller, bildebehandling, og veiplanlegging bli brukt til å generere en skanneprofil og instruere en robotarm til å utføre effektiv skanning av laksefileter, og sørge for at melaninflekker unngås?"

For å nå dette målet ble det bygd en robotstruktur som bruker Mask R-CNN til å segmentere magen og pseudo-melaninflekker på laksefilleten, etterfulgt av bildebehandling og en trinnvis planleggingsmetode inspirert av kunstige potensialfelts (KPFs) metode. Den KPF-baserte planleggeren blir brukt til å kommandere en 6-DoF robotarm i et Gazebo-simulert miljø ved å bruke rammeverket for bevegelsesplanlegging, "Moveit".

Resultatene demonstrerer en planleggingsalgoritme som er i stand til å generere komplekse planer, og tilpasse seg til flere melaninflekker og orienteringer av fileten. Den KI-drevne segmenteringen trenger noe forbedring, og bruken av pseudo-melaninflekker grunnet det utilgjengelige datasettet gjør det vanskelig å vurdere ytelsen. Den robotiserte skanningen er i stand til å nøyaktig følge den planlagte banen med en gjennomsnittlig utførelsestid på 3.365 sekunder, tilsvarende en gjennomsnittlig TCP-fart på 116.3 $mm\ s^{-1}$. Dette er innenfor den nødvendige eksponeringstiden for en akseptabel Raman skanning.

Fremtidig arbeid innenfor dette forskningsområdet inkluderer å bruke et relevant datasett for bedre trening, testing og validering av den KI-drevne modellen og KPF-baserte planleggeren. Videre, ved å teste i et laboratoriemiljø med en Raman spektroskopisensor og et bevegende samlebånd, vil robotsystemet kunne bli videre justert og optimert, som gjør det mulig å implementere løsningen i en relevant miljø.

Nomenclature

Abbreviations

AI	Artificial Intelligence	-
APF	Artificial Potential Field(s)	-
DoF	Degrees of Freedom	-
EE	End effector	-
FOV	Field of View	-
IBVS	Image-Based Visual Servoing	-
IMU	Inertial measurement unit	-
NIR	Near infrared	-
PBVS	Position-Based Visual Servoing	-
RGB-D	Red, green, blue, distance	-
ROS	Robot Operating System	-
TCP	Tool Center Point	-
UR	Universal Robots	-

Contents

Chapter 1

Introduction

This chapter will give a brief introduction to the background of the master's thesis and the motivation related to the assessment of food quality in the salmon industry. It will highlight one of the biggest problems in the field and some of the previous research aimed at overcoming it, before presenting the objectives of this thesis and the contribution.

1.1 Background

Salmon is dominating the fish industry in Norway, being the most sold slaughtered fish at approximately 1.6 million metric tonnes in 2022, equivalent to more than 102 billion NOK [1]. The salmon contains important nutrients, such as omega-3, which has numerous health benefits for consumers. The level of omega-3 fatty acids varies between salmon, and research conducted by Ytrestøyl et al. [2] show that there is a connection between high fatty acid content and a redder and, therefore, healthier appearance. This opens the door for quality sorting, where healthy-looking fillets with higher concentrations of omega-3 can be sold at a higher price, as quality is used by consumers in decision-making.

One of the bigger problems in the salmon industry is melanin spots, which have been a challenge for many years. They affect the quality perception by consumers and, therefore, also the selling price. Over the years, this problem has increased, with 7% of salmon being affected in 2003, and 20% being affected in 2015 [3]. The believed origin of this problem has been changing over the years: vaccines, impact injuries, nutrition and viruses [4]. Recent discoveries suggest the problem originates from various causes [5]. Nowadays, workers have to manually detect and remove these spots from the affected areas in the filleting line, which is a labor intensive and costly solution, and causes price reductions of 9 to 67% [6].

A well-established technique used for fatty acid estimation is chromatography, and since gas chromatography was first used to for analytical fatty acid estimation, it has been a vital technique within chemical analysis [7]. However, since it must be carried out in a laboratory, it is not suitable for production lines in the industry. To achieve this, a faster, non-destructive method is needed, and a study led by Afseth et al. [8] shows that Raman spectroscopy has potential for use in fatty acid prediction. In this research, Near-infrared (NIR) and Raman spectroscopic data were compared and evaluated to determine the quantification of fatty acids in salmon. Further studies show that Raman is also viable for in-line measurements, where scans were carried out on moving samples with short exposure (up to 4 seconds) [9].

Using computer vision to detect melanin spots is not new. Research carried out in 2007 [10] shows that a simple camera can be used to automatically detect these unwanted spots in Atlantic salmon fillets, and with the advancements of artificial intelligence (AI) over the years, this technology can be further enhanced. Studies led by Fenelon et al. [11] demonstrate the effectiveness of using robotic system to perform Raman spectroscopy on salmon fillets moving on a conveyor at 25 $mm\ s^{-1}$ in a proof of concept demonstration. The authors' tests include the use of real fillets, and demonstrate that a simple straight scan profile along the belly of the fillet can be successfully followed by a Raman sensor connected to the robot's TCP. The next steps discussed by the authors include developing a pilot-scale system solution, commanding the sensor to follow complex trajectories, and even avoiding melanin spots, to estimate fatty acid values with greater accuracy.

Enabling an efficient and automatic in-line measurement system for salmon fillets creates opportunities not only for quality assurance but also provides large datasets, which can be analyzed to get a better understanding of the problem. All of this together will help increase the quality of the salmon fillet, positively impact consumption, and expand the market.

1.2 Objective

The objective of this thesis is to develop, integrate, and test a path planning algorithm based on artificial potential fields (APF) method, that generates complex 2D paths on a static salmon, using melanin spots as obstacles. The new path planning algorithm will be denoted as an APF-like path planner. The generated path will be used to control a 6-DoF robot arm to move a Raman spectroscopy sensor across the fillet in a simulated environment. To achieve this, a combination of machine vision, AI-driven models and image processing techniques will be used to extract the intricate information about the fillet, so a path planning framework can be built to generate complex, but feasible paths for the simulated 6-DoF robot arm to follow (see Figure 1.1). The simulated system can thus be used in future implementations, allowing a real robot arm to scan natural salmon fillets. The objectives of the master's thesis are summarized in Table 1.1 below.

Table 1.1: Main objectives of the Masters' Thesis.

Objective	Description
AI-driven models	Run AI-driven models for instance segmentation
Feature Extraction	Use image processing to extract information for the path planner
Path Planner	Build a path planning algorithm for melanin spot avoidance
Simulation	Test the robotic system in simulation

Based on the background and objectives presented, the following research question was formulated, which this thesis aims to answer::

How can machine vision, AI-driven models, image processing, and path planning be used to generate a scan profile and instruct a robot arm to perform an effective scanning of salmon fillets, ensuring the avoidance of melanin spots?

1.3 Contribution

The contribution of this thesis is to extend and enhance the development of the work presented by Fenelon et al. [11]. This includes taking the robotic Raman spectroscopy system to a level in which it can use a 3D camera and AI-driven models to segment pseudo-melanin spots in a simulated environment. By building a new path planner, inspired by the APF method, which is able to generate a path using AI-based segmentation, and finally deprojecting this path from the image plane to the world

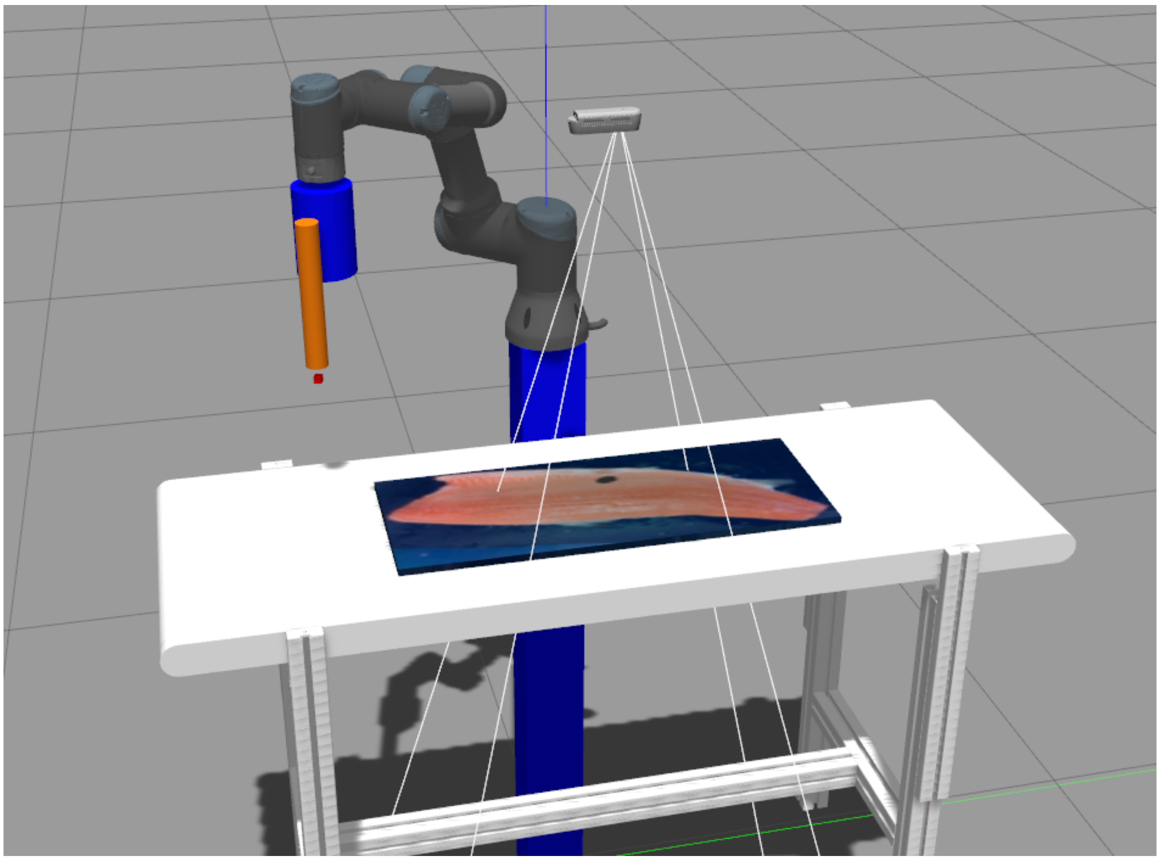


Figure 1.1: Simulated Robotic Raman Spectroscopy System.

frame, the robot arm can follow it with a dummy Raman spectroscopy sensor attached to its end effector. Simulation results are presented showing the trajectories of the robot end-effector and the joint angles for different cases of melanin spot occurrence in a given salmon fillet.

Chapter 2

Theory

This chapter will cover the necessary theory which lay the foundation for this thesis. It aims to provide a better fundamental understanding of the different theoretical concepts and topics which are explored and discussed later on this work.

The chapter begins by providing the reader with a better understanding of kinematics and frames in robotics, before exploring the field of vision systems and depth sensing. Furthermore, the general concept of control systems are explained followed by some different models of AI-learning and some common techniques used in this field. Lastly, artificial potential fields will be explained, followed by some image processing techniques.

2.1 Kinematics and Frames

The theoretical concepts and formulas which lay the foundation for kinematics in this thesis are predominantly extracted from the 'Springer Handbook of Robotics', 2016, which provides a comprehensive overview and explanation of the many topics within the field [12]. The goal of this section is to get a better understanding of robot kinematics, which is fundamental topic before more complex fields are covered. Kinematics in robotics is essentially the study of movement of rigid bodies, where the required forces to perform the motion are left aside. Since the forces required are not considered, kinematics can focus on the *pose*, that is, position and orientation, and the velocity of the body. This is because higher derivatives of position and orientation, such as linear and angular acceleration, require forces and torques to be present [12].

2.1.1 Displacement and Rotation of Poses

One prerequisite to express the pose of an object is to have another *coordinate frame*, commonly referred to as *frame* for reference. A frame consists of an origin point, denoted as O_i and three orthogonal axes ($\hat{x}_i \ \hat{y}_i \ \hat{z}_i$). This is because the pose or displacement of one frame always needs to be relative to some other frame. When a frame is considered to be moving, it is implied that the observer is stationary within another frame that is considered fixed, and the movement of the given frame is relative to the fixed frame.

To represent the position of the origin of frame i relative to frame j, a 3×1 vector is used, as seen in Equation 2.1. This vector consists of the three Cartesian coordinates of the origin of the i frame $\mathcal{F}_i$ with respect to the j frame.

$$ {}^{j}\boldsymbol{p}_i = \begin{pmatrix} {}^{j}{p_i}^{x} \\ {}^{j}{p_i}^{y} \\ {}^{j}{p_i}^{z} \end{pmatrix} . \tag{2.1} $$

When this vector is not equal to zero, there is a displacement along at least one of the axes, called a translation. This means that the object is not rotated, but simply displaced in some direction.

In addition to displacement, a frame can also be rotated. This means that the origin of the frame remains in the same position; however, when a frame is rotated, the alignment between at least one of its original x, y, or z-axis and their corresponding orientation after rotation will no longer be parallel. To represent the orientation of coordinate frame i with respect to coordinate frame j, the basis vector representation of i with respect to j is used. This yields (${}^{j}\hat{x}_i \ {}^{j}\hat{y}_i \ {}^{j}\hat{z}_i$), and when this is written as the 3×3 matrix, it is called a rotation matrix ${}^{j}\mathbf{R}_i$:

$$ {}^{j}\mathbf{R}_i = \begin{pmatrix} \hat{x}_i \cdot \hat{x}_j & \hat{y}_i \cdot \hat{x}_j & \hat{z}_i \cdot \hat{x}_j \\ \hat{x}_i \cdot \hat{y}_j & \hat{y}_i \cdot \hat{y}_j & \hat{z}_i \cdot \hat{y}_j \\ \hat{x}_i \cdot \hat{z}_j & \hat{y}_i \cdot \hat{z}_j & \hat{z}_i \cdot \hat{z}_j \end{pmatrix} . \tag{2.2} $$

The basis vectors are unit vectors, meaning that they are of norm 1, and since the dot product between two unit vectors is just the cosine of the angle between them, the formula in Equation 2.2 can be simplified for a rotation around each axis. The elementary rotations of angle θ of frame i about the z, y and x-axis of frame j can be written as:

$$\mathbf{R}_Z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{R}_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}, \quad \mathbf{R}_X(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}.$$

These matrices can also be combined through matrix multiplication. This means that if the rotation matrices ${}^{j}\mathbf{R}_i$ and ${}^{k}\mathbf{R}_j$, then the orientation of frame i relative to the frame of k can be expressed as:

$$ {}^{k}\mathbf{R}_i = {}^{k}\mathbf{R}_j \, {}^{j}\mathbf{R}_i \,. \tag{2.3}$$

The basis vectors of coordinate frame i and j are both orthonormal, meaning that each vector is a unit vector and orthogonal to the other basis vectors. The formed columns of ${}^{j}\mathbf{R}_i$ from the dot product of said vectors will also be mutually orthonormal. In mathematics, a matrix in which the vectors are mutually orthonormal and of a unit-norm 1 is an orthogonal matrix, and orthogonal matrices have a special property where the inverse and the transpose of the matrix are equal. This property provides six auxiliary relationships, being three for orthogonality and three for unit length, and as the rotation matrix consists of nine elements, only three of them are needed to represent the orientation.

So far, displacement and rotation have been explained separately, but in many cases, a frame can be both rotated and displaced, which calls for the topic of homogeneous transformations. The goal of homogeneous transformations is to combine the position vectors and rotation matrices. If the relationship between to coordinate systems, for example, i and j, is known, a vector ${}^{i}\mathbf{r}$ (expressed through the i frame) can be transformed to the j frame by combining the two notations ${}^{j}\boldsymbol{p}_i$ and ${}^{j}\mathbf{R}_i$ to describe the position of the origin and orientation of the frame, respectively. This gives the equation:

$$ {}^{j}\mathbf{r} = {}^{j}\mathbf{R}_i \, {}^{i}\mathbf{r} + {}^{j}\,\mathbf{p}_i, \tag{2.4}$$

which can be further written as:

$$\begin{pmatrix} {}^{j}\mathbf{r} \\ \mathbf{1} \end{pmatrix} = \begin{pmatrix} {}^{j}\mathbf{R}_i & {}^{j}\mathbf{p}_i \\ \mathbf{0}^T & \mathbf{1} \end{pmatrix} \begin{pmatrix} {}^{i}\mathbf{r} \\ \mathbf{1} \end{pmatrix}, \tag{2.5}$$

and, more concisely:

$${}^{j}\mathbf{T}_i = \begin{pmatrix} {}^{j}\mathbf{R}_i & {}^{j}\mathbf{p}_i \\ \mathbf{0}^T & \mathbf{1} \end{pmatrix}. \tag{2.6}$$

Equation 2.6 is the homogeneous transformation matrix, which transforms vectors from the i coordinate frame to the j coordinate frame. Its inverse, denoted as $({}^{j}\mathbf{T}_i)^{-1} = {}^{i}\mathbf{T}_j$, will transform vectors from coordinate frame j to i.

As with rotation matrices, these transformation matrices can also be combined through matrix multiplication, which means that the transformation ${}^{k}\mathbf{T}_i$ can be written as:

$${}^{k}\mathbf{T}_i = {}^{k}\mathbf{T}_j \, {}^{j}\mathbf{T}_i \,. \tag{2.7}$$

Performing a simple rotation around an axis can be denoted as **Rot**, and a translation along an axis can be denoted as **Trans**. A rotation of θ around the $\hat{z}$-axis, and the translation of d along the $\hat{x}$-axis axis can be written as:

$$\mathbf{Rot}(\hat{\mathbf{z}}, \theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2.8}$$

$$\mathbf{Trans}(\hat{\mathbf{x}}, d) = \begin{pmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.9}$$

This concludes the topic of pose representation. Being able to describe systems in 3D is fundamental for robotics applications, and it also enables several components to communicate sensor data with each other. The information from a camera is of no use to a robot arm if there is no way of transforming the data into a coordinate system in which the robot arm operates.

2.1.2 Forward Kinematics

Based on the foundations of frame transformations covered in Section 2.1.1, it is possible to describe a robot arm using the relation between the individual links and joints. The link length is essentially the displacement, and the joint angle is the rotation between two joint frames. Using the principles of transformation matrices, it is possible to describe the pose of the end effector using a series of transformation matrices. For a simple two-dimensional planar arm with two joints $\mathbf{q} = [q_1\ q_2]^T$ and two links a_1 and a_2, rotating about the z-axis, the transformation from the base frame to the end effector frame can be expressed as [13]:

$$ {}^{e}\mathbf{T}_b = \underbrace{Rot(z, q_1)}_{\text{joint 1}} \oplus \underbrace{Trans(x, a_1)}_{\text{link 1}} \oplus \underbrace{Rot(z, q_2)}_{\text{joint 2}} \oplus \underbrace{Trans(x, a_2)}_{\text{link 2}} . \qquad (2.10) $$

2.1.3 Inverse Kinematics

As shown in Section 2.1.2, it is possible to find out the pose of the end effector through the joint angles using the forward kinematic map of the arm. Another fundamental part of robot kinematics is the inverse kinematics, which is naturally the opposite of the forward kinematics. Instead of finding the pose of the end effector through the joint angles, the required joint angles to achieve a desired end effector pose are found [13].

2.2 Vision System and Depth Sensing

A robotic system does not necessarily require a vision system to work. If the task of the robot arm is repetitive, and there are no variations between instances, the movement of the arm can be programmed to strictly perform this task. One example of this is car assembly. The arm can be invoked once a prerequisite condition is satisfied, e.g., the chassis has arrived at its designated location. However, for a robotic system to operate autonomously in unknown environments or perform a task that may vary slightly or considerably due to randomness or unique instances, some kind of external sensor system is needed. An example of this is the interaction between a robot arm and heterogeneous foods on a moving conveyor belt. The food may vary slightly, and the exact position of the given food is not known. The use of vision systems for a given application enables the robot arm to act based on real-world sensor data, which can be useful in robotics systems.

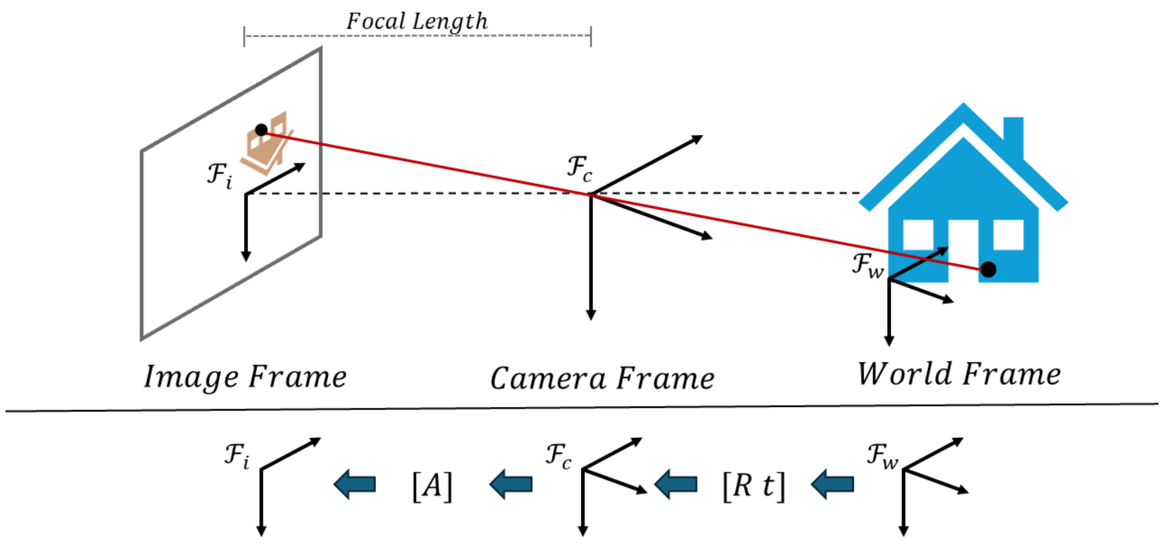


Figure 2.1: Camera Pinhole Model showing the camera frame in the middle, with the image frame to the left and world frame to the right.

2.2.1 Camera Intrinsics and Extrinsics

Most RGB-D (red, green, blue, depth) cameras can be described with the pinhole model, which describes how a three dimensional point in the world frame can be transformed into the two dimensional point in the image plane. There are two sets of parameters for the camera to account for in this transformation: the intrinsic and the extrinsic parameters.

The extrinsic parameters account for the displacement and rotation between the camera frame and the world frame while the intrinsic parameters account for the internal parameters of the camera. This includes coordinates of the principal point, which is where the optical axis intersects with the image plane, and a parameter representing skewness [14]. This concept is visualized in Figure 2.1, where the optical axis is represented with a red line from the world frame through the pinhole and projected at the image plane. The transformation from some arbitrary point $\tilde{M} = [X\ Y\ Z\ 1]^T$ in the world frame $\mathcal{F}_w$, to the corresponding pixel values $\tilde{\mathbf{m}} = [u\ v\ 1]^T$ can be described with the following formula [14]:

$$s\tilde{\mathbf{m}} = \mathbf{A}[\mathbf{R}\ \mathbf{t}]\tilde{M}\,. \tag{2.11}$$

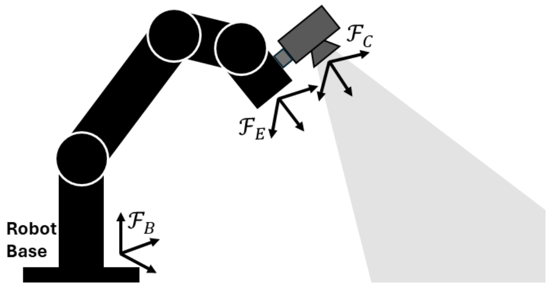
Here, s is an arbitrary scale factor and $\mathbf{A}$ is the matrix with the intrinsic parameters,

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.12}$$

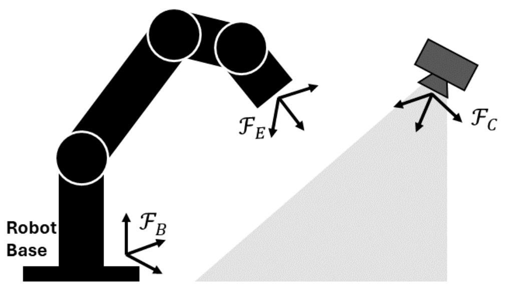
Where u_0 and v_0 represent the coordinate of the principal point, α and β are the focal lengths in pixels of the two axes u and v, and γ is a factor for describing the skewness in the image. $[\mathbf{R}\ \mathbf{t}]$ is the homogeneous transformation matrix which maps the point from the world frame to the camera frame using rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$, as discussed in Section 2.1.1. Once the intrinsic parameters are calibrated, they do not need to be updated, unlike extrinsic parameters that naturally change when the camera is moved. Of course, the intrinsic parameters will need to be updated when a new camera model is used.

2.2.2 Camera Configurations

A camera can be configured in two ways when used with a robot arm system. The two have favorable and unfavorable factors, which will be further explored in this section.



(a) Eye-in-Hand camera configuration.



(b) Eye-to-Hand camera configuration.

Figure 2.2: Camera configuration setup.

Eye-in-Hand

The first configuration, seen in Figure 2.2a, is called Eye-in-Hand. This implies that the camera is attached to the end effector of the robot, and the configuration results in a non-stationary camera with respect to the environment. However, the relationship between the camera frame $\mathcal{F}_C$ and the end effector frame $\mathcal{F}_E$ will remain constant.

This also implies the transformation matrix ${}^{E}\mathbf{T}_{C}$, is constant, and calibration is relatively simple. With this configuration, the robot itself will likely not obstruct the cameras field of view (FOV). The eye-in-hand also allows more freedom in field of view, as it is not limited to the robots workspace. If ${}^{E}T_{C} = I$, then the following relationship holds:

$$ {}^{B}\mathbf{T}_{C} = {}^{B}\mathbf{T}_{E} \,. \tag{2.13} $$

Eye-to-Hand

The second option for camera configuration is Eye-to-Hand, as seen in Figure 2.2b. In this configuration, the camera is fixed in the workspace. Therefore, it will not be a constant relationship between the robot end effector frame $\mathcal{F}_E$ and the camera frame $\mathcal{F}_C$, and the transformation ${}^{E}\mathbf{T}_{C}$ will be expressed based on the robot base frame $\mathcal{F}_B$:

$$ {}^{E}\mathbf{T}_{C} = \ {}^{E}\mathbf{T}_{B} \ {}^{B}\mathbf{T}_{C} \,. \tag{2.14} $$

This configuration requires a more complex calibration, but it also allows the camera to capture objects outside the robots workspace. It does not add extra weight to the end effector of the robot, compared to the eye-in-hand configuration.

2.2.2.1 Calibration using Internal Sensors

One method for finding the configuration error between two frames by using internal sensors is presented by Leite et al. [15]. The method defines the known frame $\mathcal{F}_b$ and the frame $\mathcal{F}_{b_1}$, and aims to find the configuration error between them, denoted by $\mathbf{x}_{bb_1} = [p_{bb_1}^T \ \Phi_{bb_1}^T]^T$. Here, p_{bb_1} denotes the translation error, and $\Phi_{bb_1} = [\phi \ \vartheta \ \psi]^T$ represent the orientation error, expressed, in roll, pitch and yaw angles, with R_{bb_1} being the corresponding rotation matrix..

The configuration error is found by using a set of $i = [1, 2, \ldots, N]$ calibration points, denoted by, $\lambda_1.\lambda_2, \ldots, \lambda_N$, whose positions with respect to frame $\mathcal{F}_b$ are known, and are estimated with respect to frame $\mathcal{F}_{b_1}$ using the following equations:

$$
\begin{aligned}
p_{b\lambda_1} &= p_{bb_1} + R_{bb_1} p_{b_1\lambda_1} \ , \\
p_{b\lambda_2} &= p_{bb_1} + R_{bb_1} p_{b_1\lambda_2} \ , \\
&\vdots \\
p_{b\lambda_N} &= p_{bb_1} + R_{bb_1} p_{b_1\lambda_N} \ ,
\end{aligned} \tag{2.15}
$$

Here, $p_{b\lambda_i} \in \mathbb{R}^3$ denote the position of the calibration point λ_i with respect to the robot base frame $\mathcal{F}_b$, and $p_{b_1\lambda_i} \in \mathbb{R}^3$ is the position of the calibration point λ_i with respect to frame $\mathcal{F}_{b_1}$, as introduced in Section 2.1.1, both being the components of the transformation matrix $T_{ij} = [R_{ij} \ p_{ij}]$ with $i = \{b, b_1\}$ and $j = \lambda_i$.

To find the homogeneous transformation matrix T_{bb_1}, the configuration of the calibration points regarding the frames $\mathcal{F}_b$ and $\mathcal{F}_{b_1}$ can used:

$$
T_{bb_1} = T_{b\lambda_i} \ T^{-1}_{b_1\lambda_i} \ , \qquad\qquad i = 1, 2, \ldots, N. \tag{2.16}
$$

The configuration error, denoted $\mathbf{x}_{bb_1} = [p^T_{bb_1} \ \Phi^T_{bb_1}]^T$, can be found through the solution of a non-linear least squares problem by using an appropriate method, like Newton's method [16]. First, an objective function f, which describes the system through the position and orientation vectors, must be defined. This is based on the stacked equations in Equation 2.15, and should be solved for $f(p_{bb_1}, R_{bb_1}) = 0$,

$$
f(p_{bb_1}, R_{bb_1}) = \begin{bmatrix} p_{b\lambda_1} - p_{bb_1} - R_{bb_1} p_{b_1\lambda_1} \\ \vdots \\ p_{b\lambda_N} - p_{bb_1} - R_{bb_1} p_{b_1\lambda_N} \end{bmatrix} \in \mathbb{R}^{3N}. \tag{2.17}
$$

In the previous equations, each of them has three position coordinates and six orientation constraints. Therefore, the minimum number of endpoints λ to perform the calibration procedure is three. However, by choosing a higher number, the effect of measurement noise is mitigated, and the configuration error accuracy is improved. Equation 2.17 is minimized by using its Jacobian J_f, given by

$$
J_f = \nabla f = \left[\frac{\partial^T f}{\partial p_{bb_1}} \ \frac{\partial^T f}{\partial \Phi_{bb_1}} \right]^T , \tag{2.18}
$$

since $\dfrac{\partial f}{\partial p_{bb_1}} = -[I \cdots I]^T \in \mathbb{R}^{3N\times 3}$, and to obtain Equation 2.18 is only necessary to calculate:

$$\frac{\partial f}{\partial \Phi_{bb_1}} = -\begin{bmatrix} \frac{\partial R_{bb_1}}{\partial \phi} p_{b_1\lambda_1} & \frac{\partial R_{bb_1}}{\partial \vartheta} p_{b_1\lambda_1} & \frac{\partial R_{bb_1}}{\partial \psi} p_{b_1\lambda_1} \\ \vdots & \vdots & \vdots \\ \frac{\partial R_{bb_1}}{\partial \phi} p_{b_1\lambda_N} & \frac{\partial R_{bb_1}}{\partial \vartheta} p_{b_1\lambda_N} & \frac{\partial R_{bb_1}}{\partial \psi} p_{b_1\lambda_N} \end{bmatrix}, \tag{2.19}$$

where $\frac{\partial R_{bb_1}}{\partial \phi}, \frac{\partial R_{bb_1}}{\partial \vartheta}, \frac{\partial R_{bb_1}}{\partial \psi} \in \mathbb{R}^{3\times 3}$ and $\frac{\partial f}{\partial \Phi_{bb_1}} \in \mathbb{R}^{3N\times 3}$.

Lastly, the correction in $\mathbf{x}_{bb_1}$ is found at each iteration k using Newton's method by

$$\mathbf{x}_{bb_1}(k+1) = \mathbf{x}_{bb_1}(k) - \beta J_f^{\dagger}(k) f(p_{bb_1}(k), R_{bb_1}(k)), \tag{2.20}$$

in which $\beta > 0$ and $J_f^{\dagger}(k) = (J_f^T(k) J_f(k))^{-1} J_f(k)^T$ represents the left pseudo-inverse of J_f.

As explained by the authors [15], the dimension of this Jacobian matrix is $3N \times 6$, and it can grow significantly larger by choosing many endpoints.

2.3 Control Systems

One prerequisite to better understand visual servoing and robot motion control is the topic of control systems in general. The goal of a control system is to maintain some process output close to a desired value by manipulating some control variable (actuators) to counter the disturbances by which the process is affected [17]. The topic of control systems is a wide and deep research area, and for this thesis, only the necessary concepts and theories will be explored.

2.3.1 Closed- and Open Loop Controllers

One common way of controlling a process is by using closed loop controllers. The principle behind this is to compute the process error e, being the difference between the desired value y_{sp} of the process output, and the actual, or measured value y_m, and use it to update the control variable (actuators) accordingly.

$$e = y_{sp} - y_m. \tag{2.21}$$

One example is temperature control in a room, where a sensor measures the error between the desired temperature and the current temperature. The error will provide information on whether the heater (actuator) should be turned up or down, but how much it should be adjusted is more complicated, and is the foundation of control systems. Other examples might include simple cruise controls in cars, water level control in tanks, and dynamic positioning systems of vessels. The key to designing a satisfactory control system is knowing how to process the different variables to achieve minimal error over time. A block diagram illustrating the process, the controller, the sensor, and common variables is shown in Figure 2.3

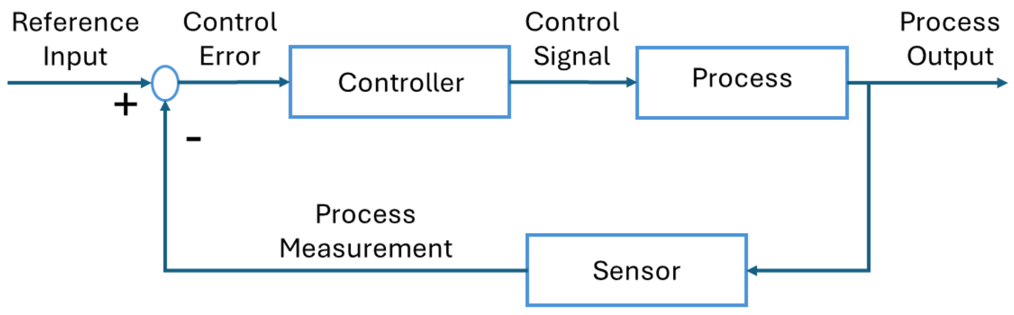


Figure 2.3: General feedback controller showing how the measurement error is used to update the control signal.

Over the years, many techniques have been developed to efficiently drive the error in a feedback system to zero. These include the popular proportional-integral-derivative (PID) controller [18], state-space controllers [19], model predictive control [20] and Lyapunov-based control [12]. The details covering the signal processing and filtering within the control module lie beyond the scope of this thesis, as the general concept of feedback loops and their structure is of greater importance.

As opposed to a closed loop system, like the feedback controller, an open loop system is also common. However, since these processes are predetermined and there is no feedback loop, the input signal(s) to the process does not depend on the system output, as shown in Figure 2.4. Examples of such processes are time-based processes like toasters, traffic lights, and electric hand dryers [21], which only operate for a given amount of time based on the input settings.

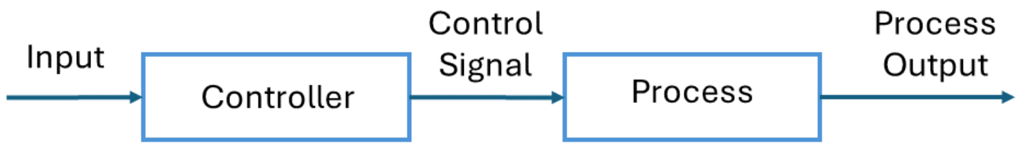


Figure 2.4: General predetermined open loop controller with no feedback to update control signal.

2.3.2 Joint Space Control

In robotics, feedback loops are commonly used to control robotic manipulators, and one common control method is the *joint space controller* [12]. As explained in section 2.3.1, a feedback controller uses the error from Equation 2.21 to update the process, which in this case is the joint angles. The general feedback loop of joint space controllers is visualized in Figure 2.5.

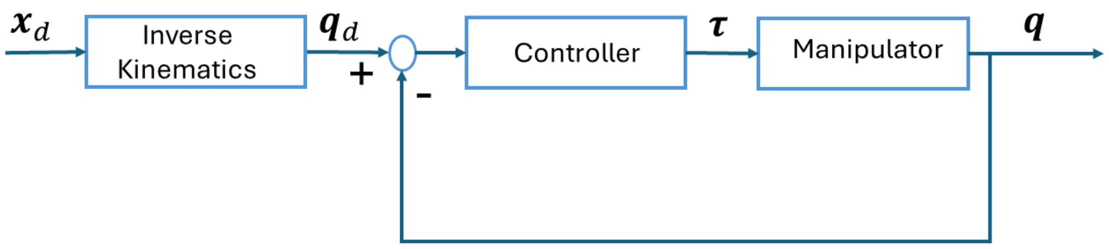


Figure 2.5: Joint space controller using the error from the actual joint angles to update the input to the manipulator.

As seen in Figure 2.5, the system first gets some desired position $\mathbf{x}_d$ of the end effector, and uses the inverse kinematics of the arm to find the corresponding angles $\mathbf{q}_d$ of the joints. A tuned controller then finds the required torques τ to achieve these angles and the robotic manipulator outputs joint angles, which are used for the feedback error. It is also worth mentioning that the inverse kinematics are calculated on some interpolated points between the current joint angles $\mathbf{q}$ and the desired joint angles $\mathbf{q}_d$ [12].

2.3.3 Visual Servoing

Visual servoing is a robotics area that covers the use of camera data to control the robot. As described in Section 2.2, this makes it possible for the robot to operate autonomously in unknown environments and perform various tasks. For this thesis, the camera, with the help of an AI-driven model, is used to retrieve data from the salmon fillet on the conveyor belt. The concepts and fundamental theory surrounding the topic of visual servoing are based on the book Robotics, Vision and Control by Peter Corke [13].

2.3.3.1 Position-based Visual Servoing

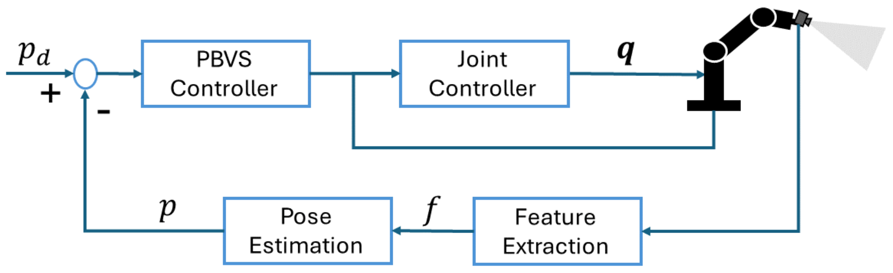


Figure 2.6: PBVS, feature extraction is used to estimate some pose, which is used as feedback error.

In a PBVS system, the pose of the goal with respect to the camera is estimated. The objective is to determine the motion required to move the camera from its initial pose to the desired pose. Since the required movement can be significantly large, it is common to split into multiple smaller fractions of the movement, essentially interpolating the error in joint angles.

If the goal pose is moved, the motion of next step will take it into consideration, as the relative pose and its fractional step is computed at each time step. PBVS requires the intrinsics and pose of the camera with respect to the robot to be known to be able to estimate the pose of an object in the image frame as explained in Section 2.2.1.

2.3.3.2 Image-based Visual Servoing

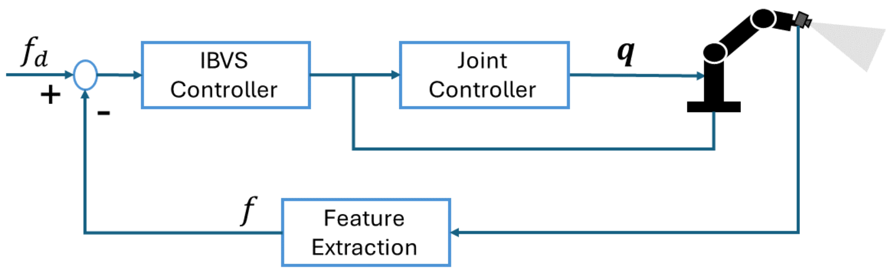


Figure 2.7: IBVS, extracted features in the observed scene are used directly as feedback error.

While PBVS tries to minimize the pose error of the goal in the control loop, image-based visual servoing (IBVS) tries to minimize the error of the visual features of the goal in the loop. Given two different camera positions, one can imagine the position of the edges, or some other feature of an object can differ considerably. This opens the door to a control loop that tries to align some feature points on an object to some desired position in the image frame. This approach will not let the observed features fall outside the camera frame.

2.4 AI-based Learning

This section will cover the necessary concepts in the field of AI-based learning, according to the book 'Python Machine Learning', Second Edition by Sebastian Raschka and Vahid Mirjalili [22]. Starting with an insight to deep learning and convolutional neural networks, before covering some common techniques used for training them. Furthermore, the different types of image classification will be covered before an introduction to model evaluation.

2.4.1 Deep Neural Networks

The aim of deep neural networks is to learn from data, so a model can be built to make predictions. A network is built by first extracting a number of features from some input data, and feeding it into a layered network of neurons, which learns by tuning weights, and outputs a prediction. Deep neural networks consist of multiple layers, which are able to see more complex patterns in the input data, but they are increasingly computationally heavy.

2.4.2 Object Detection and Instance Segmentation

Object detection and segmentation are pivotal tasks in the field of computer vision, as they play a fundamental role in applications such as autonomous driving, medical research and video surveillance. The key role of object detection or segmentation, is to retrieve spatial information of an object/area of interest in an image, video or video stream. There are four main levels of classification in this field of research. The simplest is classification, which aim to only classify some object in an image. Classification + localization takes it a step further by retrieving its spatial information with a bounding box. The third is object detection, which is able to classify instances of objects and their respective spatial information by creating bounding boxes with a corresponding label. Lastly is segmentation, which goes into more detail than the previously mentioned techniques, as it aims to classify the object in an image on a pixel-level. There are two types, instance segmentation, which separates instances of the same object, and semantic segmentation, which instead groups them as a single entity.

2.4.3 Model Evaluation

One common technique used to evaluate a model, is splitting the available data into training, testing and validation sets. The training and validation data is used during model training, while the test data is used for testing the model on unseen data. These two will give the indication of *overfitting*, or *underfitting.* Overfitting describes the behaviour when the model is performing really well on the training data, but poorly on the validation or test data. This effectively means that the model learns patterns in the training data that are not very generalized, and will not apply for new data. Underfitting, on the other hand, means that the model is not able to learn any patterns from the training data, and performs poorly in general. This requires a more complex model, and it is important to find a balance between these two cases to build a model that performs well on unseen data [22].

2.4.4 Data Augmentation

Data augmentation [23] is a technique used in AI-based learning where new image data is generated by augmenting the already available dataset. This can include skewing, stretching, flipping, adding brightness, and cropping (to name a few). The technique can make a model more generalized, and therefore avoid overfitting. It can also help balance data by generating more data of one class, for example, if a model is built to detect two different classes, like cats and dogs, but the dataset consists mostly of cat images.

2.5 Artificial Potential Fields - A Path Planning Algorithm

This section discusses some of the main aspects related to path planning algorithms for robotics. Planning a collision-free path for a robotic system is a fundamental task, and even though it may seem simple, it can be a complex problem in robotics. The thesis will take a closer look at artificial potential fields, which is an intuitive, yet complex algorithm for path planning and obstacle avoidance. The formulas and theory regarding artificial potential fields (APF) are based on material from the book 'Robotics: Modelling, Planning and Control', by Siciliano et al. [24], which provides a wide and in-depth analysis of the various topics within robotics.

Potential fields are nothing new. In fact, the approach was presented in 1986 by Oussama Khatib [25], and since then has been a popular planning algorithm in robotics. It uses an analogy for the obstacle avoidance problem. The method describes the robot as a particle moving in an artificial potential field U across a planning space $\mathcal{C}$, where the principle of gradient descent is used for navigation. The gradient descent tries to lead the robot to a global minima q_g, by traversing towards the negative gradient of the field $-\nabla U$. This negative gradient is effectively the force affecting the particle, meaning a steep configuration in U will lead to a substantial force. This force is calculated at each step, and determines the direction and size of the next step.

The potential field, and its corresponding force vector field, is defined based on the position of the goal q_g and the areas of $\mathcal{C}$ which are defined as any i present obstacles $\mathcal{CO}_i$. The current position of the robot is denoted by q, and the algorithm works by having the goal q_g exert a *global*, radial attractive force $\mathbf{F}_a(q)$, while the stationary obstacles exert a *local*, radial repulsive force $\mathbf{F}_r(q)$. The objective of the attractive force is to lead the robot towards the goal, and the repulsive fields try to keep the robot away from them. The sum of said forces:

$$\mathbf{F}_{tot}(q) = \mathbf{F}_a(q) + \mathbf{F}_r(q),$$

lay the foundation for the heterogeneous force vector field across $\mathcal{C}$.

2.5.1 Attractive Potential and Attractive Force

The attractive potential $U_a(q)$ is global across $\mathcal{C}$, and there are in general two possible approaches for defining this field, being paraboloidal and conical. A paraboloidal attractive field is defined by:

$$U_{a1}(q) = \frac{1}{2}k_a e^T(q)e(q) = \frac{1}{2}k_a||e(q)||^2, \tag{2.22}$$

where $e(q) = (q_g - q)$ and k_a is a positive tunable parameter to control the gain. The paraboloidal field implies that the resulting force, being the gradient of the potential, is proportional to $e(q)$:

$$f_{a1}(q) = -\nabla U_{a1}(q) = k_a e(q). \tag{2.23}$$

The paraboloidal function ensures that the attractive force $f_{a1}(q)$ decreases close to q_g. This helps achieve a soft landing, and not overshooting; however, it increases indefinitely with e. This can lead to poor performance in terms of obstacle avoidance far from q_g, as $f_{a1}(q) \gg f_r(q)$.

The conical attractive field ensures a constant attractive force, regardless of e. It is defined by:

$$U_{a2}(q) = k_a||e(q)||, \tag{2.24}$$

with its respective force being:

$$f_{a2}(q) = -\nabla U_a(q) = k_a \frac{e(q)}{||e(q)||}. \tag{2.25}$$

This approach avoids the problem of indefinitely scaling attractive force, but the force does not diminish as q approaches q_g, and can lead to overshooting the goal.

One way to ensure both a soft landing, and avoiding infinitely scaling potential far from q_g, is by combining the two approaches. By defining a threshold distance ρ, the new attractive field can be defined as:

$$U_a(q) = \begin{cases} \frac{1}{2}k_{a1}||e(q)||^2 & \text{if } ||e(q)|| \ \leq \ \rho\,, \\ k_{a2}||e(q)|| & \text{if } ||e(q)|| \ > \ \rho\,. \end{cases} \tag{2.26}$$

To ensure a continuous f_a function for this combined approach, the following relationship must be satisfied [26]:

$$k_{a1}e(q) = k_{a2}\frac{e(q)}{||e(q)||} \quad \text{for } ||e(q)|| = \rho\,, \tag{2.27}$$

which can be achieved by simply defining $k_{a2} = \rho k_{a1}$.

2.5.2 Repulsive Potential and Repulsive Force

Unlike the attractive potential, which has some non-zero value globally across $\mathcal{C}$, the repulsive potential is locally defined around the i present obstacles defined in $\mathcal{C}$ as $\mathcal{CO}_i$.

It is most common to define this potential field as:

$$U_{r,i}(q) = \begin{cases} \dfrac{k_{r,i}}{\gamma}\left(\dfrac{1}{\eta_i(q)} - \dfrac{1}{\eta_{0,i}}\right)^{\gamma} & \text{if } \eta_i(q) \leq \eta_{0,i}\,, \\ 0 & \text{if } \eta_i(q) > \eta_{0,i}\,, \end{cases} \tag{2.28}$$

where:

$k_{r,i} > 0$:	is the parameter for repulsive gain for obstacle i.
$\gamma = 2, 3, \ldots$:	is the exponent of the function, impacting the field behaviour.
$\eta_{0,i}$:	is the repulsive field extent of obstacle i.
$\eta_i(q)$:	is the minimal distance to obstacle i, $\eta_i(q) = \min_{q' \in \mathcal{CO}_i} \lvert\lvert q - q' \rvert\rvert$.

The repulsive potential of $\mathcal{CO}_i$ also exerts a repulsive force, being the gradient $-\nabla U_{r,i}(q)$, of the potential field.

$$f_{r,i}(q) = -\nabla U_{r,i}(q) = \begin{cases} \dfrac{k_{r,i}}{{\eta_i}^2(q)}\left(\dfrac{1}{\eta_i(q)} - \dfrac{1}{\eta_{0,i}}\right)^{\gamma-1} \nabla\eta_i(q) & \text{if } \eta_i(q) \leq \eta_{0,i}(q)\,, \\ 0 & \text{if } \eta_{0,i}(q) > \eta_{0,i}\,. \end{cases} \tag{2.29}$$

Since $\eta_i(q)$ is the minimal distance from q to the defined obstacle, the repulsive force $f_{r,i}$ is orthogonal to the field at q and directed away from q'. There can be several obstacles in $\mathcal{C}$, meaning a position q can be affected by more than one obstacle. To account for this when visualizing the potential field or finding the net force in q, the sum of the p repulsive potentials or forces must be found, respectively.

$$U_r(q) = \sum_{i=1}^{p} U_{r,i}(q). \tag{2.30}$$

Using this repulsive field, the total potential field, and thus a total force field, can be found, which is simply the sum of both the attractive and all repulsive potentials.

Total Potential Field: $$U_t(q) = U_a(q) + U_r(q)\,, \tag{2.31}$$

Total Force Field: $$f_t(q) = -\nabla U_t(q) = f_a(q) + \sum_{i=1}^{p} f_{r,i}(q)\,. \tag{2.32}$$

Challenges

As for all problem-solving approaches, they may be generally effective but can fall short in certain scenarios. As with gradient descent, the movement towards the negative gradient $-\nabla U_t(q)$, aims to reach the global minima q_g, where the gradient is zero. A problem occurs when the gradient reaches zero without being at the goal position. This is known as the *local minima problem*, which is the major challenge in machine learning using gradient descent, and subsequently for APF.

In more detail, this problem occurs when the sum of forces $f_t(q) \to 0$, resulting in a stagnation of the steps towards the goal. In machine learning, this problem can be solved by adding a momentum term based on previous steps to the gradient descent, which lets the optimization search escape these local minima. However, for this application, this is not necessarily viable, as the main goal is to avoid the obstacle, and not overcome, i.e., traverse over them.

To better understand a solution, it is necessary to understand when these local minima problems occur in APF. There are mainly two scenarios. One with the robot being trapped between two obstacles, where their repulsive sum equals the negative attractive force, $\sum_{i=1}^{n} f_{r,i}(q) = -f_a(q) = 0$. The other scenario is when the robot, the obstacle, and the goal form a symmetrical straight line, meaning the repulsive force vector $f_r(q)$ that occurs as the robot approaches $\mathcal{CO}_i$ will be the opposite direction as $f_a(q)$. This repulsive force will grow until it becomes the same magnitude as the attractive force, and their sum will be zero, resulting in a net zero force. The two scenarios can be seen in Figure 2.8.

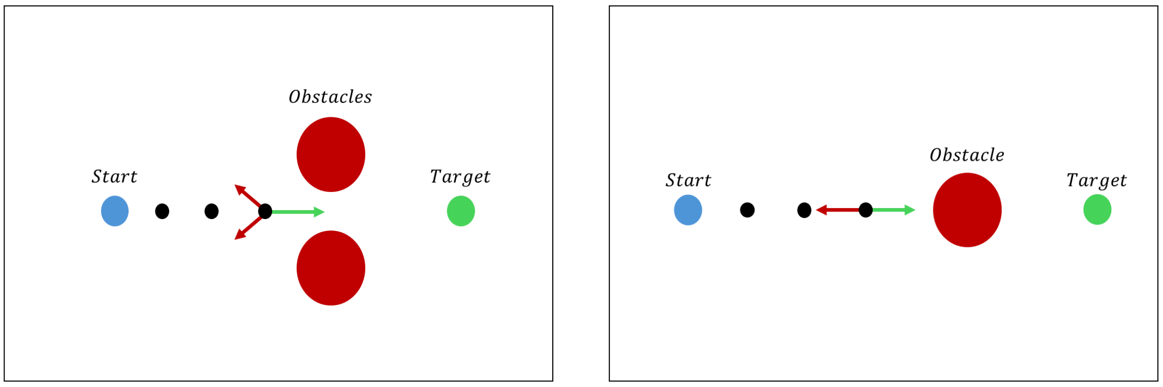


Figure 2.8: Two different ways local minima problems can occur in potential artificial potential fields.

The basic artificial potential field algorithm has no solution to this problem, other than tuning the parameters to minimize the occurrences of them. This is also the reason why the majority of papers covering the application of this method to mobile robots are focusing on overcoming the local minima problem [27]. The algorithm, however, can quite easily be modified. There are several ways of overcoming this local minima problem, ranging from simple and intuitive solutions to more complex ones. This includes vortex field surrounding the obstacles [28], wall-following mode when encountering local minima [29], and movement prediction for setting temporary goals to avoid local minima [30].

Chapter 3

Methodology

In this chapter, the methodology of this thesis will be presented. As discussed in 1.2, the goal is to develop a system capable of conducting robotic Raman spectroscopy with enhanced sensor coverage, effectively avoiding the illumination of unwanted melanin spots. This is achieved with the help of computer vision, AI-based models, image processing, and the development of a path planner based on artificial potential fields. To achieve this, a robotic pipeline is built, consisting of different components and software aimed to extract the necessary information from the raw camera feed to perform the robotic scan. This robotic pipeline is visualized in Figure 3.1.

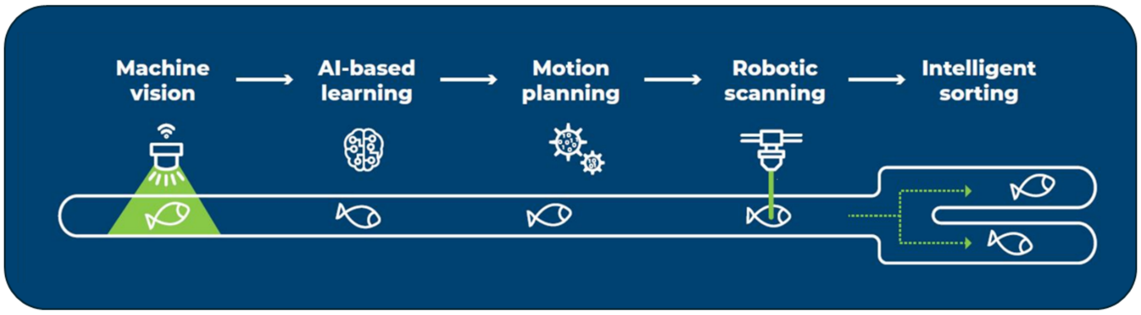


Figure 3.1: Robotic Pipeline [31]

3.1 System Overview

The entire system is built using ROS (Robot Operating System) [32], which is a widely used open-source framework for robotic applications. It is helpful for building anything from simple to more complex systems consisting of several software components or programs in C++ and/or Python that need to run and communicate with each other simultaneously.

ROS is built around an architecture using *nodes*, responsible for executing a specific function of a system, and using communicating information, structured as *messages* to *topics*. These messages are usually structured in a predefined format, decided by the type of message. They can also be custom-created by the user. The nodes can subscribe and publish to these topics to receive and send these messages respectively. ROS also has other features, as it is a deep, intricate system capable of many operations. However, for the work in this thesis, mainly publishers and subscribers are used. A simple example of this structure is shown in Figure 3.2, where two nodes are communicating with each other through a topic. The left node publishes a string-message containing "Hello World" to the topic called "talker". The right node subscribes to the topic and receives the published message. The sample code in C++ and Python can be found in the ROS tutorials [33].

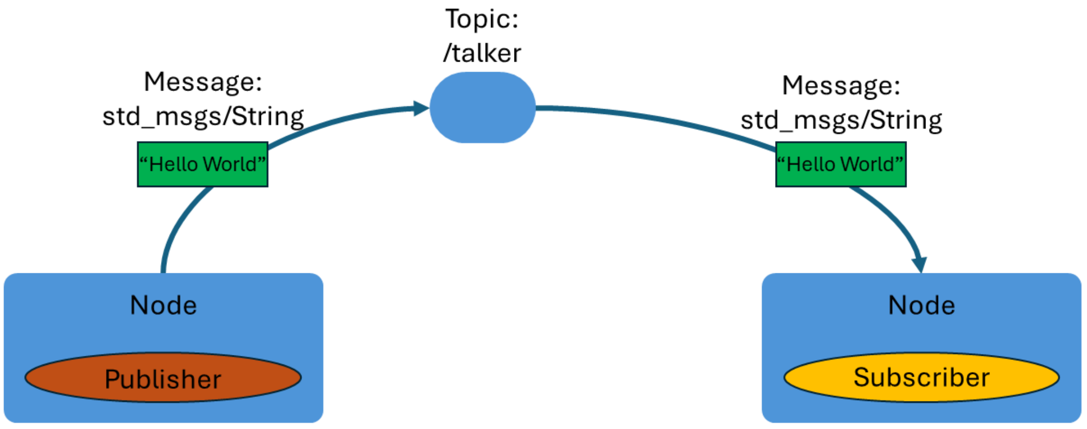


Figure 3.2: Simple visualization of ROS architecture, showing the communication between two nodes through a topic.

The system is simulated using Gazebo [34], which is a simulation tool which allows for accurate implementation of simulated sensors, advanced 3D graphics, and accurate physics and collision. It is also well integrated with ROS, making it a fitting choice for simulation. In addition, the system is visualized in Rviz, which is a visualization tool in ROS. The visualization makes it possible to see relevant information, such as frames, markers, camera feed, 3D models, as well as sensor data, like range, FOV, point clouds, etc.

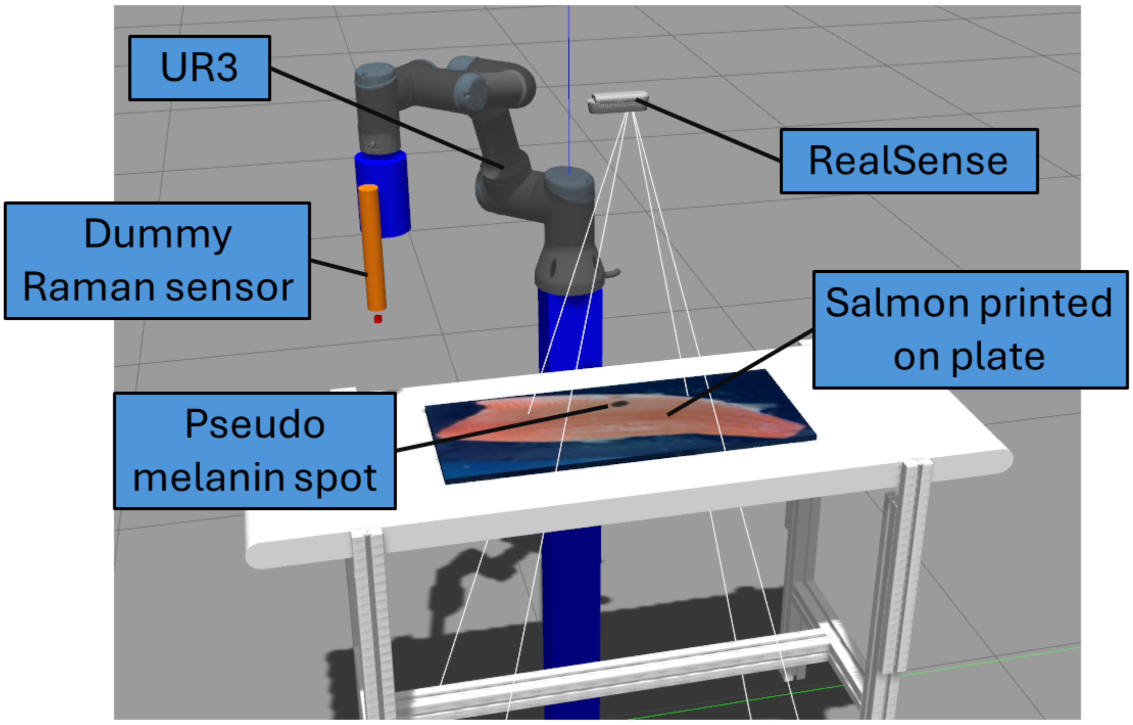
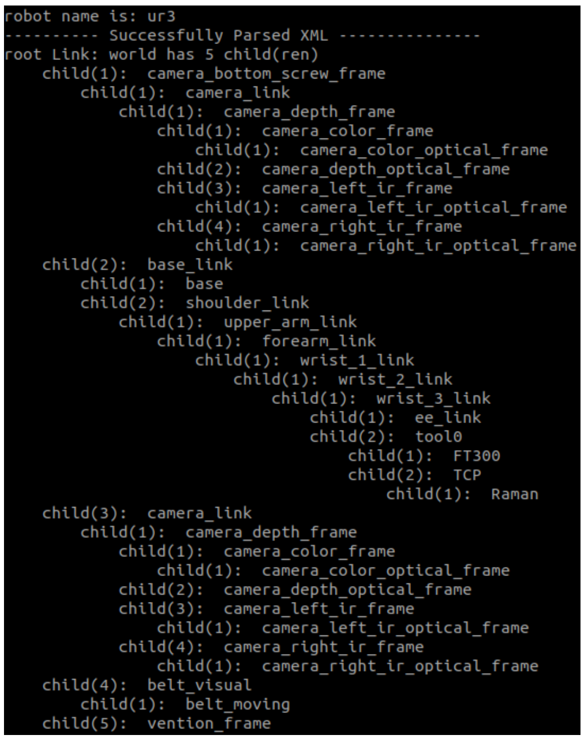


Figure 3.3: Simulated environment showing the different models loaded into Gazebo

An overview of the robotic Raman Spectroscopy system is presented in Figure 3.3, where the UR3 robot arm, a support stand, conveyor, the RealSense and a salmon are created using a URDF chain [35]. Each component in the system is considered as a link, which refers to a parent link; the world; whose origin in Gazebo is $(x = 0m, y = 0m, z = 0m)$. The robot base is mounted on a flat surface of a support structure (vention link), which is 0.1m x 0.1m x 1.050m in dimension. The vention link is fixed to the world at $(x = 0m, y = 0m, z = 1.050/2m)$, thus the robot base is placed in the world at $(x = 0m, y = 0m, z = 1.050m)$. The complete URDF chain can be seen in Figure 3.4.



```
robot name is: ur3
---------- Successfully Parsed XML ---------------
root Link: world has 5 child(ren)
    child(1):  camera_bottom_screw_frame
        child(1):  camera_link
            child(1):  camera_depth_frame
                child(1):  camera_color_frame
                    child(1):  camera_color_optical_frame
                child(2):  camera_depth_optical_frame
                child(3):  camera_left_ir_frame
                    child(1):  camera_left_ir_optical_frame
                child(4):  camera_right_ir_frame
                    child(1):  camera_right_ir_optical_frame
    child(2):  base_link
        child(1):  base
        child(2):  shoulder_link
            child(1):  upper_arm_link
                child(1):  forearm_link
                    child(1):  wrist_1_link
                        child(1):  wrist_2_link
                            child(1):  wrist_3_link
                                child(1):  ee_link
                                child(2):  tool0
                                    child(1):  FT300
                                    child(2):  TCP
                                        child(1):  Raman
    child(3):  camera_link
        child(1):  camera_depth_frame
            child(1):  camera_color_frame
                child(1):  camera_color_optical_frame
            child(2):  camera_depth_optical_frame
            child(3):  camera_left_ir_frame
                child(1):  camera_left_ir_optical_frame
            child(4):  camera_right_ir_frame
                child(1):  camera_right_ir_optical_frame
    child(4):  belt_visual
        child(1):  belt_moving
    child(5):  vention_frame
```

Figure 3.4: URDF Chain of the Gazebo Simulated Environment

The conveyor belt featured in this project comes from an existing open-source Gazebo plugin, *gazebo_conveyor* [36]. The conveyor is capable of moving, and the linear velocity can be changed by using the *max_velocity* parameter, and the power can be changed by calling a ROS service, as explained in the documentation. However, the purpose of this thesis is to develop a path planner for a static salmon fillets, i.e., a salmon fillet on a tabletop or a non-moving conveyor. The path planning and robotic scanning for a moving salmon fillet on a conveyor is considered future work. Therefore, the conveyor power is set to 0, using the ROS service,

rosservice call /robot_ns/conveyor/control "power: 0.0".

The camera is located above the conveyor, at $(x = 0.4m, y = 0m, z = 1.5m)$ with respect to the world frame $\mathcal{F}_w$, and with Z axis of the camera color optical frame pointing into the salmon. The image of the salmon fillet is imported into Gazebo by printing it onto the top surface of a relatively flat plate of dimensions (0.6m x 0.24m x 0.01m) by using the 3D modeling software Blender [37]. The different fillets were then imported by changing the image printed on this plate, which was then placed on top of the conveyor. This 2D version of the salmon fillet should be sufficient, since the camera is only considering the 2D color image of the fillet, and not the depth information.

The Flowchart in Figure 3.5 describes the software nodes/programs that connect different modules of the pipeline to achieve the objective.

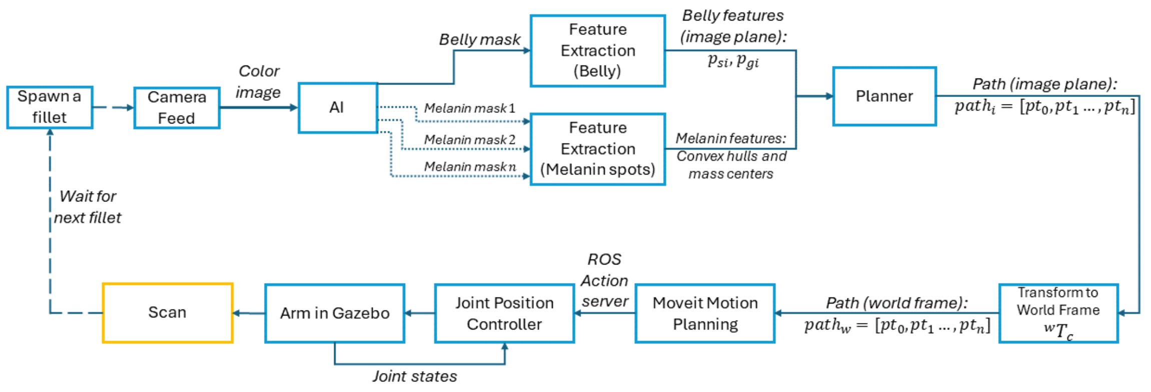

Spawn a fillet
Camera Feed
Color image
AI
Belly mask
Melanin mask 1
Melanin mask 2
Melanin mask n
Feature Extraction (Belly)
Feature Extraction (Melanin spots)
Belly features (image plane): p_{si}, p_{gi}
Melanin features: Convex hulls and mass centers
Planner
Path (image plane): $path_i = [pt_0, pt_1 \dots, pt_n]$
Transform to World Frame ${}^{w}T_c$
Path (world frame): $path_w = [pt_0, pt_1 \dots, pt_n]$
Moveit Motion Planning
ROS Action server
Joint Position Controller
Arm in Gazebo
Joint states
Scan
Wait for next fillet

Figure 3.5: System Flowchart

The system pipeline begins with a static salmon which is printed on a plate being spawned in Gazebo. The user can define its position and orientation, but for it to lie on the top surface of the conveyor, it should be spawned at $+0.3m$ in x, $\pm 0.3m$ in y, and $+1.0m$ in z, all with respect to the world frame $\mathcal{F}_w$.

The camera in the pipeline captures a complete view of a salmon fillet. The camera feed consists of ROS topics (RGB and Depth information/images) that are used as the input to the AI system. Details regarding the camera are explained in Section 3.2: Machine Vision.

The AI module consists of a Mask R-CNN model using Detectron2 to extract binary images (called masks) of the belly and melanin spots using instance segmentation. The output of this module is a belly mask and melanin masks that is used to compute the path in the image plane. The details of the AI module, such as model, annotation and image augmentation are explained in section 3.3: AI Learning.

The feature extraction module uses the belly and melanin masks to compute the start and end points of a path while using the melanin spots as obstacles. OpenCV functions and algorithms such as *findContour(), convexHull(), fitLine(), and moments()*, to name a few, are used. The output of this module is the pixel values of the two ends of the belly, as well as information about the melanin spots. The details are explained in section 3.4: Feature Extraction.

The APF-like path planner uses the information from the feature extraction module to generate a series of points in the image plane. The details of the path planner are presented and explained in Section 3.5.2: APF-like path planner.

The generated points from the planner are then transformed to the world frame by using a transformation matrix. The information on this is found in Section 3.2: Eye-To-Hand Calibration.

By using the Moveit motion planning framework, the planned path is used to compute a trajectory for the UR3 robot arm. This framework has access to the information about the Gazebo simulated UR3 robot arm, and uses a joint position controller to command the arm in Gazebo using a ROS node. This is further explained in Section 3.6: Robotic Scanning.

3.2 Machine Vision

For depth cameras, the Intel RealSense [38] cameras feature high resolution and accurate depth measurements suitable for robotics applications. Their RealSense models differ from each other mostly in terms of range, FOV, shutter technology, and extra features.

Table 3.1: Intel RealSense Model Specs [38].

Model	Ideal Range	FOV (w × h)	Shutter Type	Extra
D457	0.6 m to 6 m	$87° \times 58°$	Global	GMSL, IMU, IP65
D455	0.6 m to 6 m	$87° \times 58°$	Global	IMU
D435	0.3 m to 3 m	$87° \times 58°$	Global	-
D415	0.5 m to 3 m	$65° \times 40°$	Rolling	-
D405	0.07 m to 0.5 m	$87° \times 58°$	Global	Close range camera

As seen in Table 3.1, there are a variety of models to choose from in the RealSense lineup. However, the cameras with IMU (Inertial measurement unit) will be excessive for this application, as the eye-to-hand configuration implies a stationary camera with no use of IMU. The D415 camera is a low-cost camera with a narrow FOV, in addition to having a rolling shutter, meaning the image can be distorted when capturing high-speed objects [39].

The D435 camera acts as the middle ground, featuring a wider field of view at $87° \times 58°$, suitable for the application, as it can detect objects earlier than a camera with a narrower FOV like the D415. Even though a rolling shutter would have been sufficient, as the salmon does not move a very high speeds, the global shutter acts as a bonus. The camera is also mounted within the ideal range, making it a fitting choice.

The machine vision is composed of an eye-to-hand mounted Intel® RealSense D435 RGB-D camera [40]. It is positioned on a tripod over the conveyor belt to obtain a complete view of the salmon fillet, which is about 0.6m x 0.25m x 0.03m in size. The same position applies for the simulated camera, and it uses the Realsense Gazebo plugin [41]. The camera color optical frame can be seen in Figure 3.6, and with respect to the image plane, it can be seen that the z-axis points outward, while the x- and y-axis points to the right and down respectively.

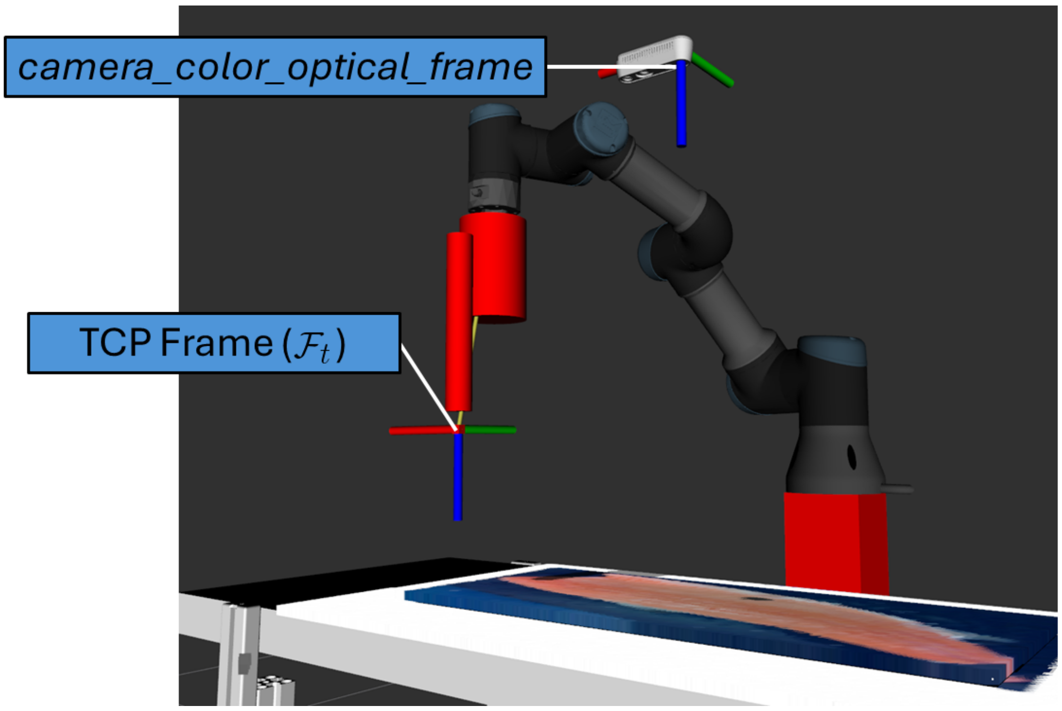


Figure 3.6: Pose of the *camera_ color_ optical_frame* above the conveyor in the simulated environment.

Camera Calibration

As explained in Section 2.2, a camera must be calibrated, both in terms of intrinsics and extrinsics. With a real camera, this can be done using the *camera_ calibration()* ROS package [42], using a printed checkerboard. This calibration utilizes the OpenCV camera calibration [14], to calculate the intrinsics and extrinsics parameters described in Section 2.2.1 until the checkerboard has no distortion.

For the simulated system, the RealSense Gazebo plugin is used [41]. This plugin makes it possible to publish information from the Gazebo simulation, like color images, depth images, and point clouds to different topics. For this application however, only the color and depth image are subscribed to by another node, so that the data can be further processed in the robotic pipeline. Note that the color and depth images are not aligned by default, so they have to be aligned by using another ROS node [43]. This node subscribes to the topics published by the color image (*/color/camera_ info*, */color/image_ raw*) and depth image (*/depth/camera_ info* and */depth/image_ raw*).

Eye-To-Hand Calibration

It is important to establish the relationship between the different frames used in the system, as it enables data to be translated between them. The relationship between the camera frame and the TCP frame, found through the eye-to-hand calibration, is what allows for the visual servoing as discussed in Section 2.3.3. To do this, *WhyCode* markers [44] can be used, which is a precise localization system utilizing small markers to estimate position. One marker is placed on the robot's end effector, enabling the RealSense camera to estimate its pose with high accuracy. There are essentially four different frames to consider, being:

- $\mathcal{F}_c$: Camera frame.
- $\mathcal{F}_b$: Robot base frame.
- $\mathcal{F}_b$: WhyCode marker frame.
- $\mathcal{F}_t$: Attached Raman sensor (TCP) frame.

These are visualized in Figure 3.7:

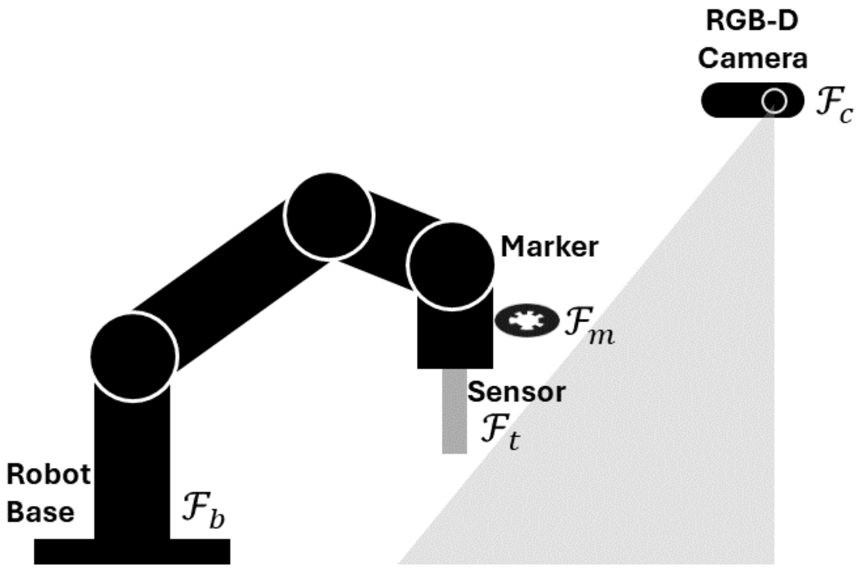


Figure 3.7: Different Frames of Robotic System

As described in Section 2.2.2, internal sensors in the robot arms joints can be used to estimate the transformation between two frames. This also applies when finding the transformation from $\mathcal{F}_b$ to $\mathcal{F}_c$. Since the marker is attached to the arms end effector, the relationship between this frame $\mathcal{F}_m$ and the tool frame $\mathcal{F}_t$ remains constant. Since the position of the end-effector is known with respect to $\mathcal{F}_b$ through the forward kinematics, the transformation can be found by solving the following set of equations, as used in experiments carried out by Fenelon et al. [11]:

$$\begin{aligned}
p_{bt}(\lambda_1) &= p_{bc} + R_{bc}p_{cm} - R_{bt}(\lambda_1)p_{tm},\\
p_{bt}(\lambda_2) &= p_{bc} + R_{bc}p_{cm} - R_{bt}(\lambda_2)p_{tm},\\
&\vdots\\
p_{bt}(\lambda_N) &= p_{bc} + R_{bc}p_{cm} - R_{bt}(\lambda_N)p_{tm},
\end{aligned}$$

This can be optimized using the optimization library from *SciPy* [45].

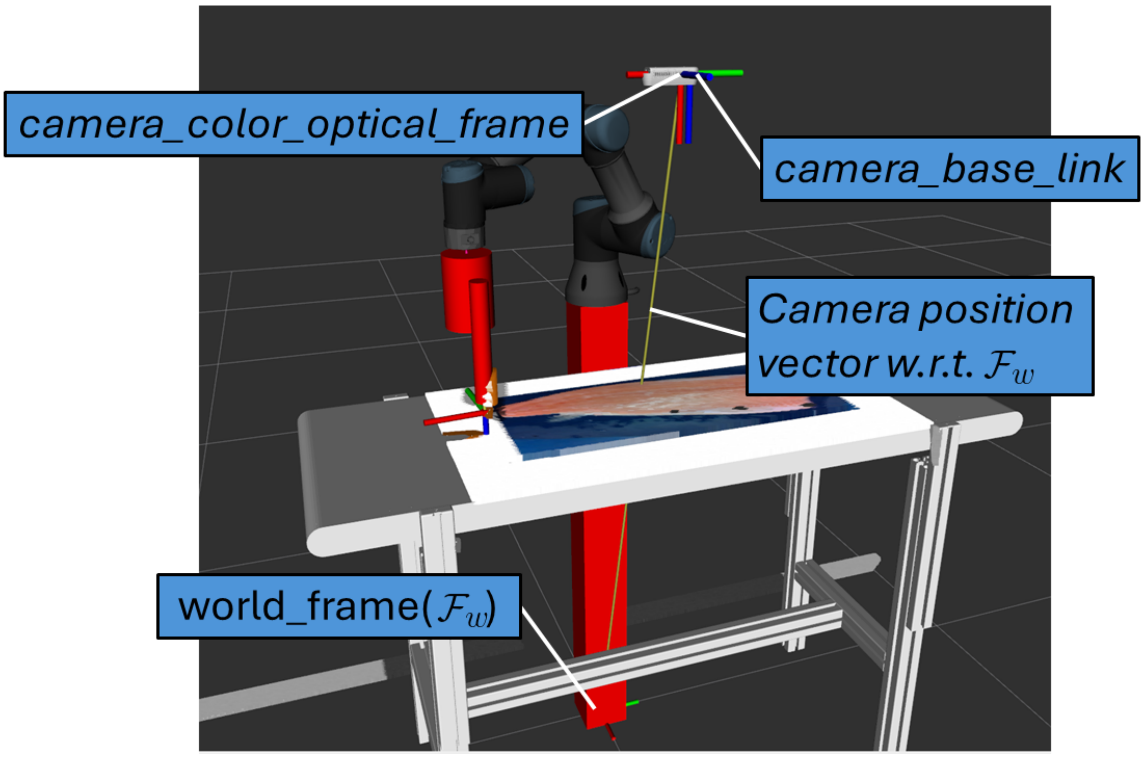

camera_color_optical_frame
camera_base_link
Camera position vector w.r.t. $\mathcal{F}_w$
world_frame($\mathcal{F}_w$)

Figure 3.8: Transformation from world frame to *camera_color_optical_frame*, $\mathbf{T}_{wc}$.

In the simulation, the arm is mounted on the stand, therefore, the hand to eye calibration is performed from the world to the camera frame, instead of the base to the camera frame. This camera position vector, as seen in 3.8, can be found using the *tf* library [46]. The URDF of the RealSense contains many links and thus results in respective frames when loaded in Gazebo and Rviz.

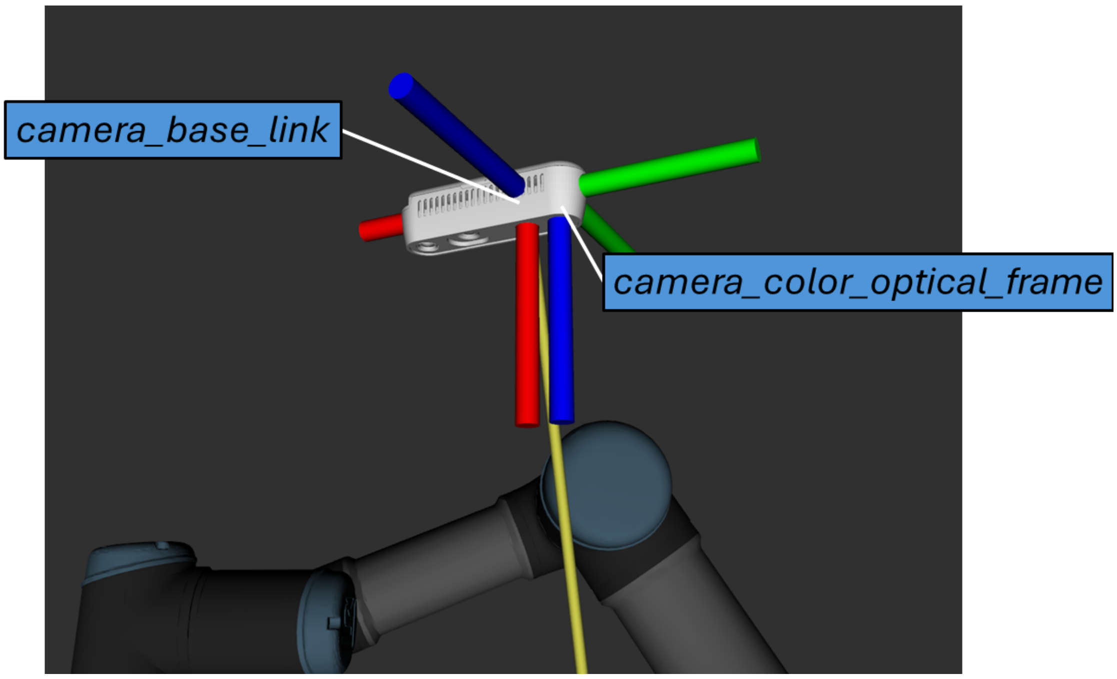


Figure 3.9: Displacement between the two frames, *camera_ color_ optical_ frame* and *camera_ base_ link*.

A yaml file is used to load the placement of the camera into Gazbo and Rviz, and its values are used to position the *camera_ base_ link* with respect to the world are as follows:

x: 0.4 m
y: 0.0 m
z: 1.5 m
$roll$: 0 $radian$
$pitch$: -3.141592 $radian$
yaw: 1.570796 $radian$

However, since the pixel deprojections are computed with respect to the *camera_ color_ optical_ frame*, as seen in Figure 3.9, the homogeneous transformation matrix from this frame to the world frame is the one of interest. Using the *tf* library, this can be computed as follows,

$tfBuffer = tf.TransformListener()$

$(trans, rot) = tfBuffer.lookupTransform(world\_frame, camera\_color\_optical\_frame, time = 0)$

$T\_world\_camera = tfBuffer.fromTranslationRotation(trans, rot)$

Here, *rot* represents the rotation, and *trans* represents the translation of the *camera_ color_ optical_frame* with respect to the world frame. Given the rotation and translation from the *lookupTransform()*, the 4x4 homogeneous transformation matrix can be computed using the *fromTranslationRotation()*, as explained in Section 2.1.1. The resulting matrix is seen in Figure 3.2.

Table 3.2: Transformation matrix from world to *camera_ optical_frame.*

$$T\_world\_camera = \begin{bmatrix} 0 & 1 & 0 & 0.4 \\ 1 & 0 & 0 & 0.015 \\ 0 & 0 & 1 & 1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It can be seen in the matrix, that the optical frame has a $0.015m$ displacement in the world frame y direction. This small displacement is clearly seen in Figure 3.9, and is between the physical frame of the camera and the lens used for the color imager.

3.2.1 Pose Estimation

The RealSense is used to deproject a generic point in the image plane, $p_{vi} = \{x_{vi}, y_{vi}\}$, to the world frame $\mathcal{F}_w$. This deprojection from the point (x_{vi}, y_{vi}) in the 2D image plane, to the corresponding 3D position $\tilde{p}_{wi}$ is calculated with the following functions from the ROS-Gazebo Realsense plugin [41]:

$$d = aligned\_depth\_frame[x_{vi}, y_{vi}] \tag{3.1}$$

$$p_{ci} = rs2\_deproject\_pixel\_to\_point(K, p_{vi}, d) \tag{3.2}$$

$$\tilde{p}_{wi} = mat\_mul(T_{wc}, \tilde{p}_{ci}) \tag{3.3}$$

where:

d	Depth measurement in meters
K	Intrinsics parameters
p_{ci}	Pose of point $\mathcal{P}_i$ with respect to camera frame $\mathcal{F}_c$
p_{wi}	Pose of point $\mathcal{P}_i$ with respect to world frame $\mathcal{F}_w$
T_{wc}	Homogeneous transformation matrix from frame $\mathcal{F}_w$ to $\mathcal{F}_c$

3.3 AI Learning

To retrieve useful information from the RealSense camera, it is necessary to implement some image processing algorithm or machine learning model. The objective of the AI step in the robotic pipeline is to detect the regions of interest, being the belly and any present melanin spots on the salmon. To achieve this, a fast real-time deep learning model for classification is used.

3.3.1 AI Model Selection

Since the goal of the AI model is to detect the belly of the fillets and any melanin spots, it is fitting to choose a segmentation algorithm. Compared to an object detection algorithm, segmentation will more precisely classify the belly, with its pixel-wise classification, compared to object detection, which only returns a bounding box. It is also not desirable to define, for example, circular-shaped melanin spots as boxes, which makes a segmentation method fitting. Instance segmentation is chosen over semantic segmentation, so that the feature extraction of the melanin masks is returned in separate binary masks, instead of having to use image processing techniques to distinguish between them.

The AI-based learning is developed with the help of Detectron2 [47], which is an open-source library of state-of-the-art machine learning models for object detection and segmentation developed by Facebook AI Research (FAIR). Their library has a wide variety of models. Examples include Faster R-CNN, RetinaNet, and RPN & Fast R-CNN. However, they are suitable for different purposes. For instance segmentation, Mask R-CNN is the only available model. The different variations available in the library are based on three different baselines, listed in Table 3.3. These are using different backbone combinations.

Table 3.3 shows that the R50-FPN model has the fastest inference time of 0.043s, which is suitable for real-time application. It also has the fastest training time per iteration. As described in the documentation [47], most of the models are trained with

Table 3.3: Detectron2 Instance Mask R-CNN models [47].

Name	lr sched	Train Time (s/iter)	Inference Time (s)	Model ID
R50-C4	1x	0.584	0.110	137259246
R50-DC5	1x	0.471	0.076	137260150
R50-FPN	1x	0.261	0.043	137260431
R50-C4	3x	0.575	0.111	137849525
R50-DC5	3x	0.470	0.076	137849551
R50-FPN	3x	0.261	0.043	137849600
R101-C4	3x	0.652	0.145	138363239
R101-DC5	3x	0.545	0.092	138363294
R101-FPN	3x	0.340	0.056	138205316
X101-FPN	3x	0.690	0.103	139653917

a 3x learning rate schedule, and that the 1x models are under-trained, but included for other purposes. This makes the R50-FPN (Model ID: 137849600) a fitting model choice for the application.

3.3.2 Artificial Melanin Spots and Image Annotation

At the time of this Thesis' production and publication, the dataset of salmon fillets with melanin spots was still not available, as it is sensitive information for the company that requires adherence to confidentiality agreements. This process took longer than anticipated, due to unforeseen events. Therefore, a previously collected dataset with healthy fillets was altered by adding pseudo-melanin spots, mimicking real melanin spots. The size of these is inspired by other research addressing melanin spots [48] [49]. The image manipulation was done in Pinta [50], an open-source program for image editing and drawing. The dataset used contains images of B-trimmed [51] salmon fillets with a resolution of 1920x1080. The images were annotated to provide the ground truth for the AI using VIA (VGG Image Annotator) [52], as seen in Figure 3.10.

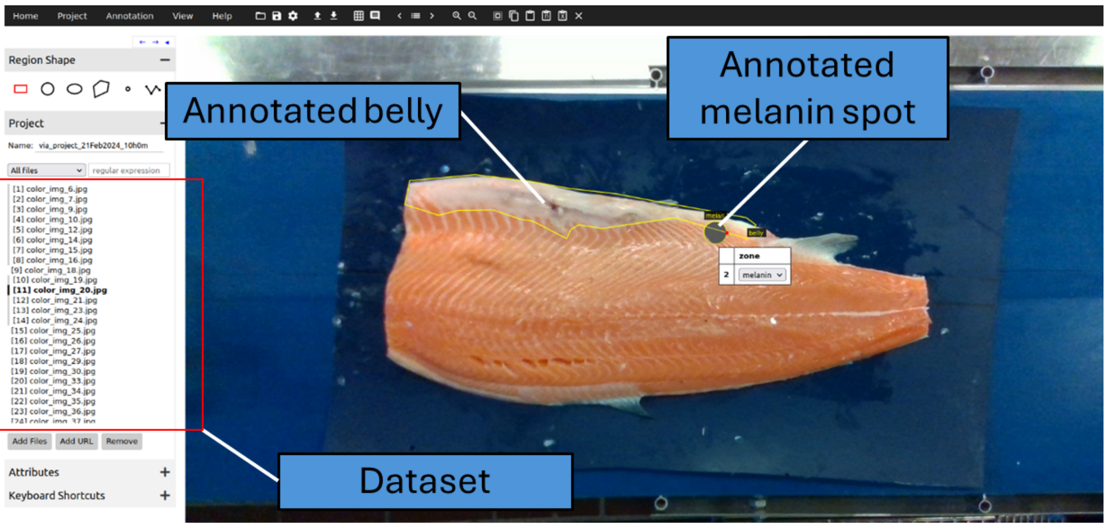


Figure 3.10: Image Annotation Using VGG Image Annotator

3.3.3 Model Training

The Mask R-CNN model was used for both detecting the belly, and the pseudo-melanin spots. The model is pre-trained on the MS-COCO dataset [53].

The model training for this thesis was done on the NMBU Robotics lab workstation with the following configurations:

Motherboard:	Lenovo P920
CPU:	Intel® Xeon(R) Silver 4114 CPU @ 2.20 GHz $\times$ 20
GPU card:	NVIDIA GeForce RTX 2080
CUDA Toolkit:	11.4
Operative System:	Ubuntu 20.04 LTS

The Detectron2 framework operates under the libraries PyTorch 1.9.0, Python 3.8 and OpenCV 4.5.6, and the ROS framework runs on the Noetic distribution that supports Python 3.

The dataset used for building the model consists of 87 images of 36 salmon, where 29 of the images has added pseudo-melanin spots. These 87 images are split in a 79:17:4 (train:validation:test) ratio, for the training of the Mask R-CNN model from the Model Zoo and Baselines [47]. Since the dataset is sparse, data augmentation is used to provide the model with more data for its training. The augmentations, value, and probability are listed in Table 3.4, and some examples of image augmentation are seen in Figure 3.11.

Table 3.4: Image Augmentation, values, and probabilities.

Transform parameter	Value	Probability
Random brightness	0.2 to 0.8	0.5
Random Contrast	0.2 to 1.5	0.5
Random Flip	horizontal	0.5
Random Flip	vertical	0.5
Random Saturation	0.3 to 1.8	0.5
Random Lightining	0.8	0.5
Random Rotation	-30 deg to 30	0.5
Random Resize	480 x 640	0.5

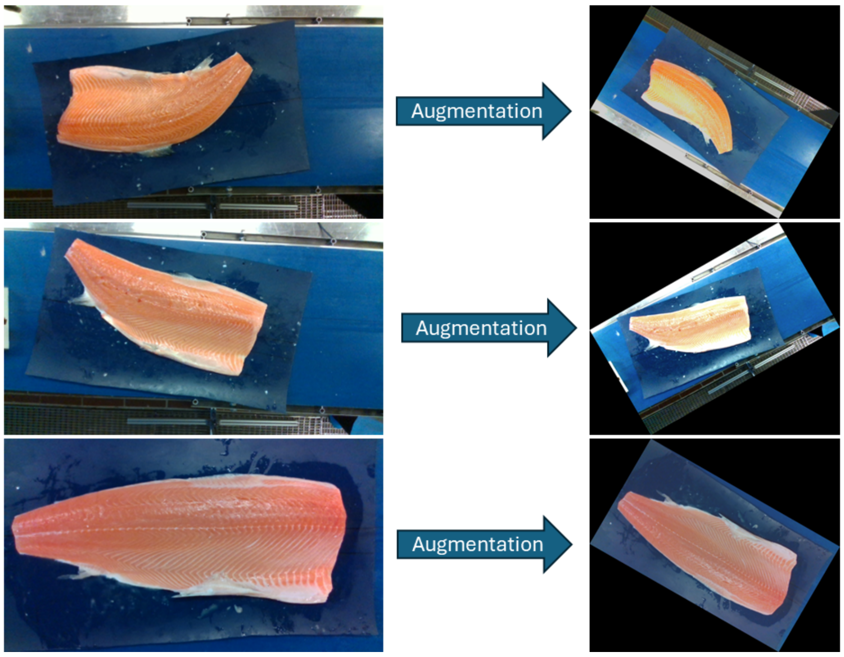


Figure 3.11: Image Augmentation Examples, showing how three different images are being applied different augmentation.

3.4 Feature Extraction

As discussed in Section 2.5, the artificial potential field model defines a field U with a known goal and start positions, being q_s and q_g respectively, as well as i known obstacle areas $\mathcal{CO}_i$. The feature extraction step aims to extract this information from the binary images provided by the machine learning model. These binary images are processed using various image processing techniques, provided by the open-source computer vision software OpenCV [54], which will be further explained in this section.

The feature extraction is essentially two processes run in parallel, where the binary image of the belly is processed to obtain q_s and q_g, and the n binary image(s) of the n melanin spot(s) are processed individually to find $\mathcal{CO}_i$. For a fillet with only one melanin spot, this is visualized in Figure 3.12.

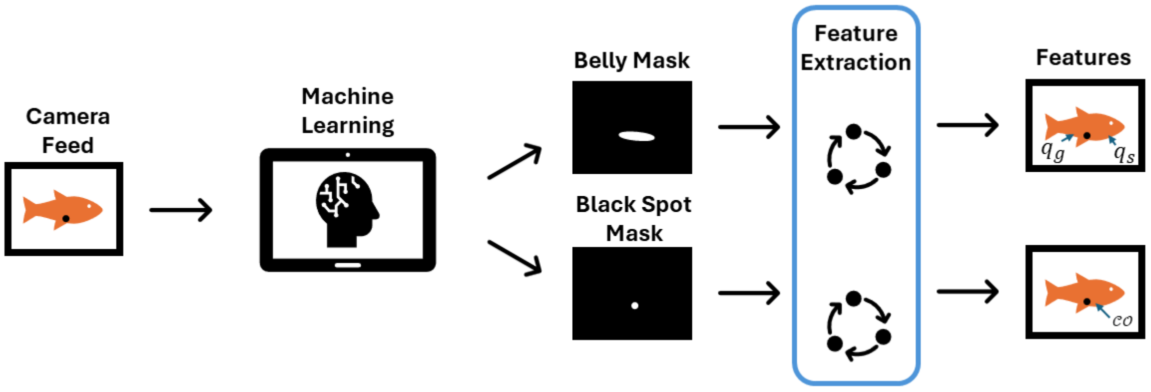


Figure 3.12: Feature Extraction visualized, showing how the camera feed is processed.

3.4.1 Extracting Features of Belly (q_s and q_g)

The desired scan line is a straight line between each of the far ends of the belly, with an added offset towards the salmon loin, as suggested by Nofima Researchers [11]. To find these points, the binary image of the belly is processed by first finding the contour of the mask, using the functions *threshold()* and *findContours()*. Using this contour, a line is fitted through it by using the function *fitLine()*, which aims to minimize the following function [55]:

$$\sum_i \rho(r_i),$$

where:

r_i:	Distance from the ith point and the line.
$\rho(r) = \frac{r^2}{2}$:	Chosen L2 distance function.

This line is constructed based on an initial point in the middle of the contour, and its slope. Based on this initial point, two close, neighboring points on the line are found on each side of this point by using elementary trigonometry. These points are then individually moved away from the initial point until they intersect with the contour. Lastly, the suggested offset toward the loin is applied, by moving the points orthogonally away from the line, a distance of $1/30$ of the scan length $||q_s - q_g||$, marking q_s and q_g.

These steps are visualized in Figure 3.13, showing a zoomed-in area of a fillet with the explained feature extraction steps for the belly.

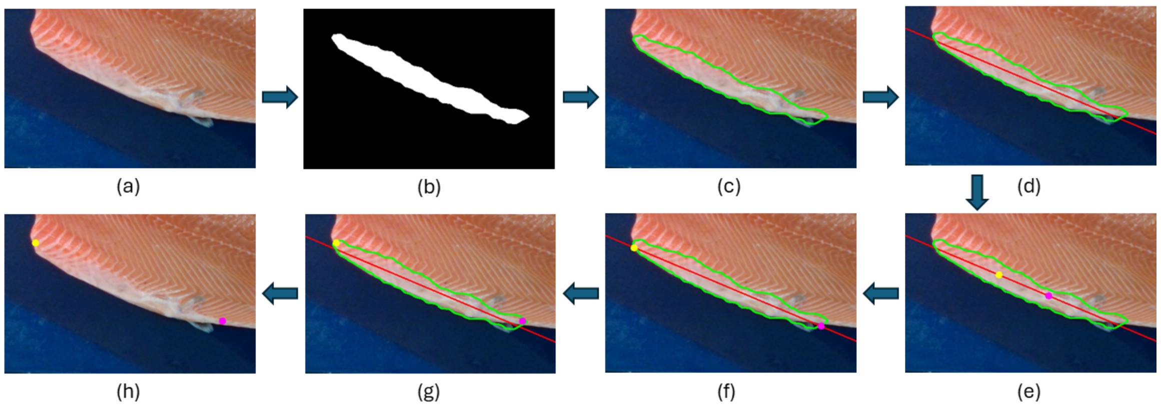


Figure 3.13: (a): Raw Image, (b): Binary Mask, (c): Contour of Mask, (d): Fitted Tine Through Contour, (e): Neighbor Points, (f): Line-Contour Intersection, (g): Added Offset, (h): Final q_s and q_g

3.4.2 Extracting Melanin Spots

Many of the same functions used to find the q_s and q_g in Section 3.4.1, will also be used extract the information about melanin spot(s). Since the AI model outputs n binary images, the presented feature extraction will apply for each individual binary image.

Defining the Area $\mathcal{CO}$

As discussed in Section 2.5, about artificial potential fields, the method can be prone to the local minima problem. One of the scenarios where local minima occur, as shown in Figure 2.8, is when the robot gets stuck in a net zero part of $\mathcal{C}$. Therefore, an approach of enclosing potential net-zero areas is chosen. This is done by first finding melanin spot i's contour, before using its corresponding convex hull to define $\mathcal{CO}_i$. This will only have an effect on concave melanin spots, and make the definition of $\mathcal{CO}$ better for the planner. It is achieved by finding the contour, followed by applying the OpenCV function *convexHull()* on the contour.

Finding the Center of Mass

In addition to the convex hull, the center of mass for a given melanin spot is found, which will be used and will be used for path planning later. The center of mass can be found using the OpenCV function *moments()* [55] on the defined convex hull, along with some simple calculations as shown in Algorithm 1.

Algorithm 1 Mass center of convex hull

```
1: $moments = cv2.moments(contour)$
2: $cx = int(moments['m10']/moments['m00'])$
3: $cy = int(moments['m01']/moments['m00'])$
4: $mass\_center = (cx, cy)$
```

Scaling $\mathcal{CO}$

Finally, to define a margin around the melanin spot, the size of this convex hull is scaled by a factor k_{scale}. This will be further explained in Section 3.5.2. This scaling is presented in Algorithm 2.

Algorithm 2 Scaling contour

```
1: $moments = cv2.moments(contour)$
2: $cx = int(moments['m10']/moments['m00'])$
3: $cy = int(moments['m01']/moments['m00'])$
4: $contour\_norm = contour - [cx, cy]$
5: $scaled\_contour = contour * k_{scale}$
```

A more visual explanation of the feature extraction of $\mathcal{CO}$ is shown in Figure 3.14, where (a) shows an arbitrary melanin spot, (b) shows the calculated contour in green, (c) shows the convex hull, as well as the mass center, and (d) shows the scaled convex hull.

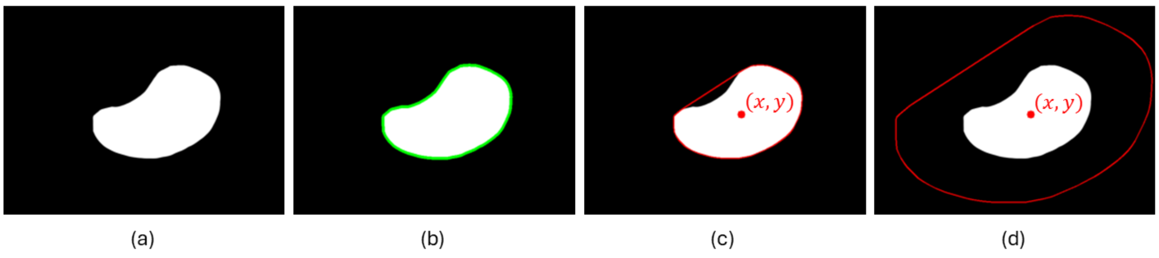


Figure 3.14: (a): Arbitrary Melanin Spot Mask, (b): Contour (green), (c): Convex Hull and Mass Center, (d): Scaled Convex Hull with Mass Center.

3.5 Path Planning

The path planner developed in this thesis is based on artificial potential fields, a planner explained in greater detail in Section 2.5. To stay consistent and to avoid misunderstandings, the melanin spots will henceforth be referred to as "obstacles", as it is more suitable when discussing the planner. As shown in the feature extraction step, the input parameters to the planner are the start- and stop positions, namely q_s and q_g, as well as the i defined obstacles $\mathcal{CO}_i$. The goal of the planner is to generate the scan profile from q_s to q_g, and avoid any obstacles that interfere with the desired straight scan profile. The planner operates in the two-dimensional image space, thus $\mathcal{C} \in R^2$. Lastly, this two-dimensional path in the image plane must be transformed to the corresponding three-dimensional path in the robots' frame with the use of the RGB-D camera, as explained in Section 3.2.1.

3.5.1 Standard Artificial Potential Fields

As discussed in Section 2.5, the bare bones APF algorithm defines an attractive function and a repulsive function, with two types of attractive potential, paraboloidal and conical. Using a paraboloidal field will result in a greater attractive force, and thus steps toward q_g, close to the starting position q_s, where the distance $||e||$ is significantly bigger. This also makes the obstacle avoidance behaviour inconsistent by having different step sizes throughout the scan line. Therefore, a conical approach was chosen, resulting in constant attraction and more predictable steps towards the goal.

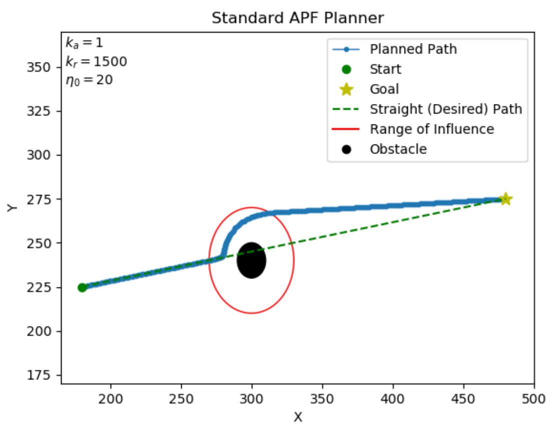
Using the formulas for attractive force with conical field, $F_{a2}(q)$ (Equation 2.25) and repulsive force $f_{r,i}(q)$ (Equation 2.29) from Section 2.5.

$$f_{r,i}(q) = -\nabla U_{r,i}(q) = \begin{cases} \frac{k_{r,i}}{{\eta_i}^2(q)} \left(\frac{1}{\eta_i(q)} - \frac{1}{\eta_{0,i}} \right)^{\gamma - 1} \nabla \eta_i(q) & \text{if } \eta_i(q) \le \eta_{0,i}(q) \\ 0 & \text{if } \eta_{0,i}(q) > \eta_{0,i} \end{cases},$$

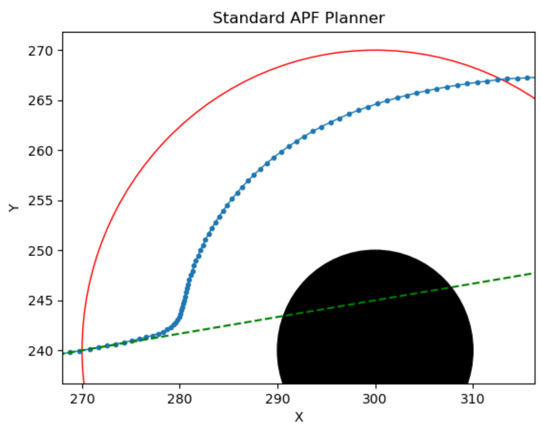
and

$$f_{a2}(q) = -\nabla U_a(q) = k_a \frac{e(q)}{||e(q)||}.$$

The resulting path is seen in Figure 3.15.



(a) Shortest path is taken



(b) Dense waypoints

Figure 3.15: Resulting Path of Normal Artificial Potential Field

Shortcomings

After running a demonstration of the bare bones APF, it occurs that some challenges arise with this planner. The planner follows the desired straight scan line (green dotted line in Figure 3.15) at the start, between $x = 180$ and $x \approx 270$. However, after it has deviated from this line at $x \approx 280$ to avoid the obstacle, it does not go back to the desired path as seen between $x = 310$ and $x = 480$. This behaviour is not unexpected from the planner, as it is designed to move towards the goal in a straight path when not obstructed by a repulsive field, but it is not optimal for generating a scan profile, as the straight line should be followed when possible. In addition to this, the planner first takes decreasingly smaller steps at $x \approx 280$, followed by larger steps at $x \approx 300$, as seen in Figure 3.15b. This is because k_r and k_a first oppose each other and then collaborate. If the planner is not tuned well, it will also have produce an oscillating path at $x \approx 285$, since the attractive and repulsive forces alternate. By instructing the robot arm to follow this path, it will have to follow unnecessarily many waypoints in this area, and it will not be able to follow the desired path after a circumvention.

3.5.2 APF-like Path Planner

The shortcomings of the basic version artificial potential field planner call for another approach aimed at overcoming the problems observed in Section 3.5.1. Mainly, being able to generate a path which follows the desired straight path when possible and also avoids compressed steps and spread steps due to opposing and collaborating forces, respectively.

Desired Behaviour:

- Follow the desired straight line when not obstructed by obstacles.
- Avoid opposing forces, resulting in clumped waypoints.
- Avoid collaborating forces, resulting in spread waypoints.
- Predictive obstacle avoidance behaviour regardless of distance $||e||$ to goal q_g, requiring homogeneous f_a.

The conventional fields are radial, which makes the attractive and repulsive forces oppose each other in $\mathcal{CO}$. This is the origin of local minima, and the varying step sizes due to opposing and collaborating forces. To avoid this problem, and create a planning algorithm that fulfills the criteria listed above, the radial fields are changed to directional-uniform, local fields, aimed at generating the necessary forces for controlled and successful planning, as well as not having the ineffective opposing forces.

The three forces used are directed either parallel with- or orthogonal to, the desired straight line from q_s to q_g. The attractive force $f_{a,goal}$, is leading the step-wise planner to q_g, is globally defined across $\mathcal{C}$, and is parallel to this desired line. It is therefore not dependent on the current configuration q, and it is effectively the only necessary force when no obstacles are present, as it will not deviate from this desired line in such case. Using the same reasoning as for the standard APF, this force is based on a conical potential field, resulting in constant, predictable steps. The field is defined as follows:

$$f_{a,goal} = k_{a,goal}\frac{j}{||j||}, \tag{3.4}$$

where $k_{a,goal}$ is the attractive coefficient, and $j = (q_g - q_s)$.

Since this attractive field is parallel to the desired path, the planner will fail to reach the goal once it first deviates from it, like in the case of circumventing an obstacle. Therefore, it is necessary to define a field that aims to take the planner back to the desired scan path after the deviation. To do this, another field, $f_{a,desired}$, is defined, which is directed orthogonally to the desired path from the current configuration q. Choosing some arbitrary points, q_s (green), q_g (star), and q (purple) this orthogonal vector, denoted w, is visualized in Figure 3.16.

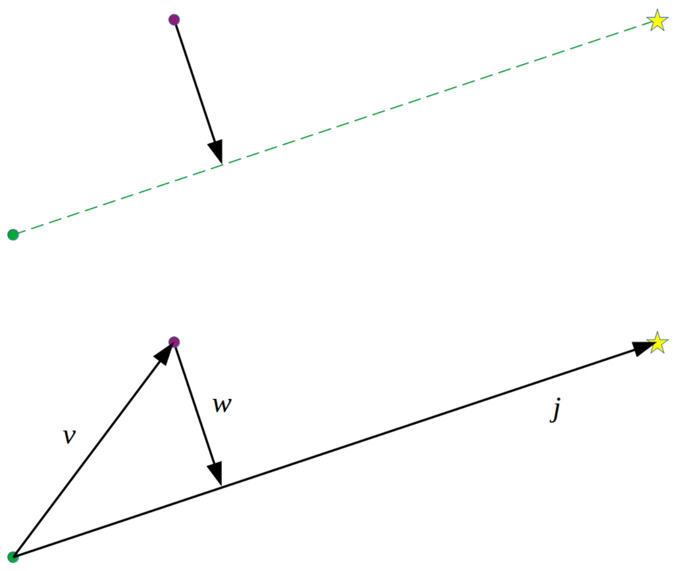


Figure 3.16: Direction of $f_{a,desired}$

Knowing $j = (q_g - q_s)$ and $v = (q - q_s)$, w can be found using the projection and simple vector calculations:

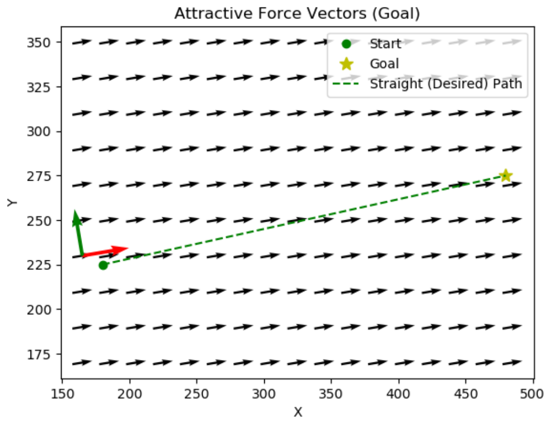
$$w = v - \frac{v \cdot j}{j \cdot j} \cdot j. \tag{3.5}$$

The field is then defined as follows,

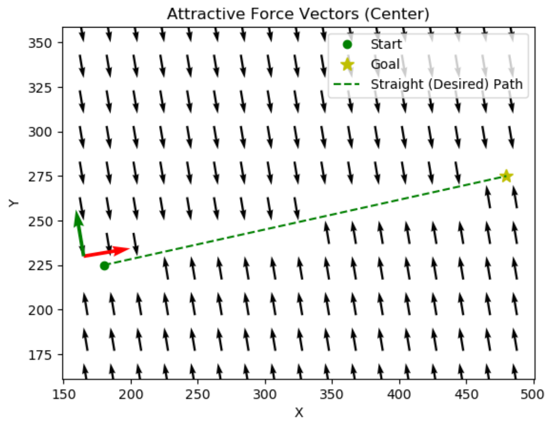
$$f_{a,desired}(q) = \begin{cases} k_{a,desired} \dfrac{w}{||w||} & \text{if } q \notin \mathcal{CO} \\ 0 & \text{if } q \ \in \mathcal{CO} \end{cases}, \tag{3.6}$$

where $k_{a,goal}$ is the attractive coefficient.

These two simple fields combined will take the planner toward q_g, and encourage it to follow the desired path, instead of the shortest path, as seen in the standard artificial potential fields. The two vector fields $f_{a,goal}$ and $f_{a,desired}$ are visualized in Figure 3.17.



(a) Attractive Field Towards Goal, $f_{a,goal}$.



(b) Attractive Field Towards the desired path (center) $f_{a,desired}(q)$.

Figure 3.17: Visualization of The Two Attractive Fields.

For the i obstacles in $\mathcal{C}$, The repulsive field in $\mathcal{CO}_i$, is defined based on its size. As explained in Section 3.4.2, the center of mass for the individual melanin spot masks are found, and is used to define the repulsive field as follows:

1. Draw a straight line inside $\mathcal{CO}_i$, through the center of mass, and parallel to the desired path.
2. For any point inside $\mathcal{CO}_i$, the field is directed orthogonally away from this drawn line.
3. The field strength is proportional to the distance to the edge of $\mathcal{CO}_i$, and the repulsive coefficient k_r

The strength of the field is given by Equation 3.7,

$$f_r(q) = \begin{cases} k_r * d_c \dfrac{-w}{||w||} & \text{if } q \in \mathcal{CO}_i \\ 0 & \text{if } q \notin \mathcal{CO}_i \end{cases}, \tag{3.7}$$

where k_r is the repulsive coefficient, w is the same as from Equation 3.5, and d_c is the distance from q, to the edge of $\mathcal{CO}_i$ in the direction of the field.

An illustrated example of a simple circumvention, and the forces affecting the planner steps is shown in Figure 3.18, where $f_{a,goal}$ is yellow, $f_{a,desired}$ is blue, and f_r is red.

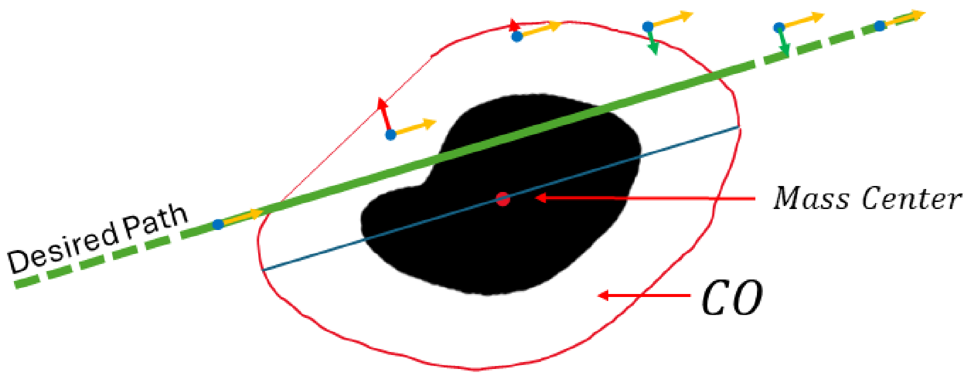


Figure 3.18: Simple Visualization of forces during circumvention, where the planned path is created from left to right.

Planner Pseudocode

Algorithm 3 Planner

```
1: current_position = q_start
2: path_image = [current_position]
3: while path < max_iterations do
4:     f_goal = $f_{a,goal}$
5:     f_desired = $f_{a,desired}$(current_position)
6:     f_repulsive = $f_r$(current_position)
7:     f_total = f_goal + f_desired + f_repulsive
8:     next_position = current_position + f_total        ▷ Take step in direction of f_total
9:     if distance(next_position, q_goal) < goal_threshold then
10:        add next_position to path
11:        add q_goal to path
12:        exit loop                                      ▷ Exit if inside goal threshold
13:    end if
14:    add next_position to path_image
15:    current_position = next_position
16: end while
17: return path_image                                     ▷ Return list of waypoints
```

The planner returns a list of waypoints $path\_image = [pt_0, pt_1, ...pt_N]$ in the image plane, which is passed forward to the robotic scanning.

3.6 Robotic Scanning

The physical robotic arm is of the type Universal Robots™CB series UR3 [56]. It has an ad hoc mounted casing for a MarqMetrix All-in-One Raman System [57]. The digital twin of the system in Gazebo features the arm [58], with a dummy Raman sensor, a conveyor belt [36], the RealSense D435 camera [41], and the image of the salmon is printed onto a flat plate on top of the conveyor.

The arm is manipulated using Moveit [59], which is state-of-the-art software for robot arm manipulation. It has packages for over 150 different robots [60], and allows for easier manipulation of both real and simulated robotic arms. An example of the Moveit interactive planner is seen in Figure 3.19.

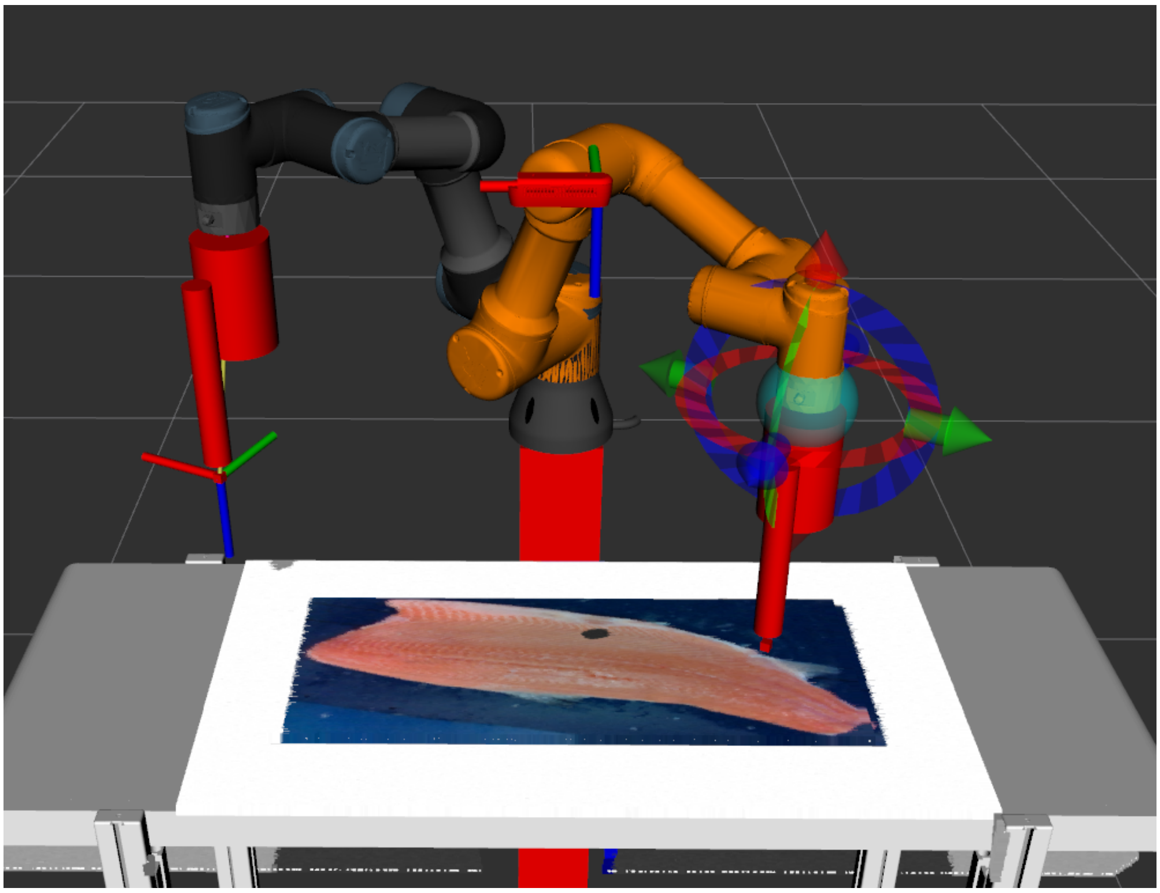


Figure 3.19: Example of the Interactive Interface of Moveit

Moveit is integrated with ROS using the *move_group* node [61], which can access information on the arm, including joint states, transform information, controller interface and more. It is used for calculating inverse kinematics. The node uses the following parameters,

planning frame:	World
joint tolerances:	0.001 radian
position tolerances:	0.05m
end_effector link:	TCP
planner:	RRT Connect
velocity, acceleration scaling:	1.0
step_size:	0.01m
jump threshold:	0.0m

The APF-like planner generates waypoints in the image plane, which needs to be transformed into a $pose\_list\_world = [pose_{w0}, pose_{w1}, ...pose_{wN}]$, in the world frame $\mathcal{F}_w$ using Equation 3.1-3.3, as repeated under:

$$d = aligned\_depth\_frame[pt_{ix}, pt_{iy}]$$
$$p_{ci} = rs2\_deproject\_pixel\_to\_point(K, pt_i, d)$$
$$\tilde{p}_{wi} = mat\_mul(T_{wc}, \tilde{p}_{ci})$$

The list of waypoints in the image plane, computed by the planner, can be lengthy, as the planner needs sufficiently small steps for the potential fields to have an effect. However, the robotic system does not benefit from a large number of waypoints, as the inverse kinematics and corresponding trajectory will have to take minor unnecessary details into consideration. To reduce the number of waypoints that have to be transformed to the world frame for the robot to follow, only a subset,n_steps, of evenly spread steps, including the start and goal, are chosen. This can be found using the open-source numerical computing library NumPy [62], as follows,

$$list\_indices = np.round(np.linspace(0, len(path) - 1, n\_steps).astype(int))$$
$$path\_with\_35\_steps = path[list\_indices]$$

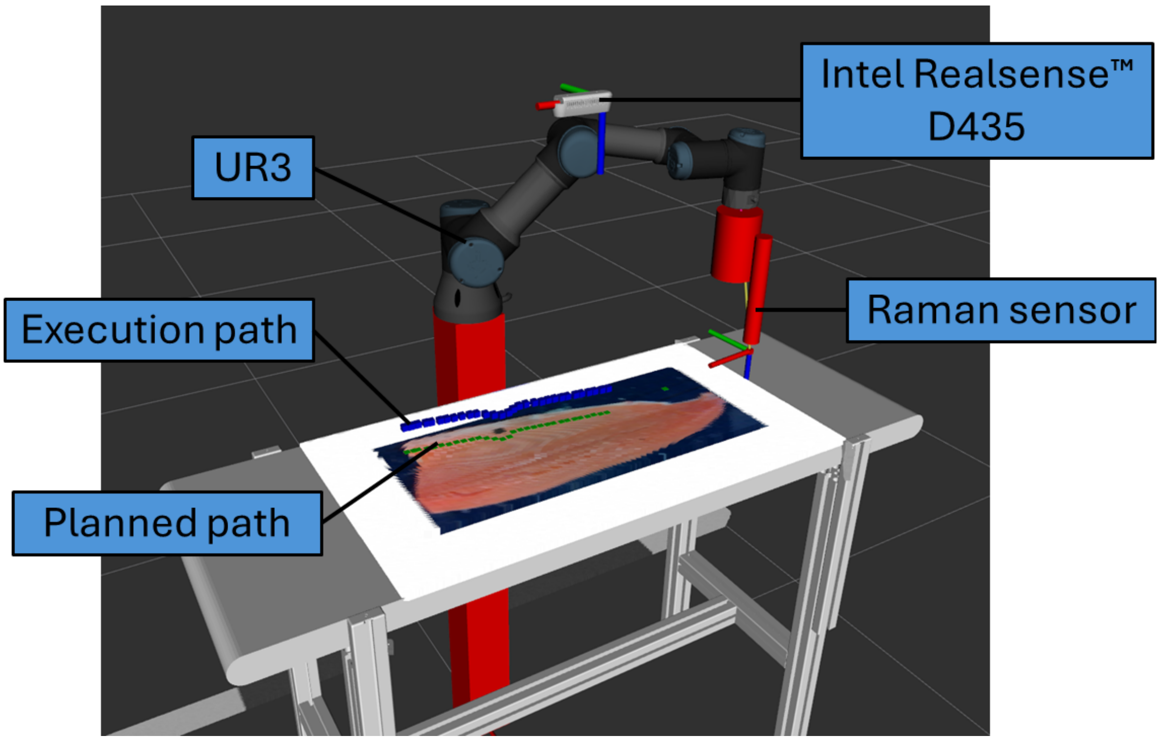


Figure 3.20: Visualized offset between deprojected path and planned path, Rviz

The computed path in the image plane and the corresponding poses in the world frame $\mathcal{F}_w$ are visualized in Figure 3.20. It is worth mentioning that the deprojected waypoints have an added z-offset from the conveyor to have some distance from the Raman sensor to the fillet, and that the orientation of the TCP is predefined, so that the sensor is orthogonal to the conveyor.

Robot Movement Types

This new list of poses $pose\_list\_world = [pose_{w0}, pose_{w1}, ...pose_{wN}]$, in the world frame is used in the *move_group* node for creating trajectories and moving the arm so that the TCP follows the desired path. Three different ways of generating a path and moving the arm have been tested, as presented by Algorithms 4-6 below.

Algorithm 4, *move_L()*

This robot movement type iterates through the list of poses by computing and executing the trajectory between each individual pose. This will make the arm move in a straight line between the poses, and since each trajectory is unique, the arm will accelerate and decelerate throughout the iteration, likely making it slow. To highlight the impact of the robot arm's acceleration and deceleration between the points, the velocity and acceleration scaling (*vel_scale* and *acc_scale*) are tested for the max value 1.0 (Algorithm 4a), and smaller value 0.2 (Algorithm 4b).

Algorithm 4 *move_L(target_poses)*

```
1: for pose in target_poses do
2:     (plan, fraction) = move_group.compute_cartesian_path(pose)
3:     plan = move_group.retime_trajectory(move_group.get_current_state(),plan,
       vel_scale, acc_scale)
4:     move_group.execute(plan)
5: end for
```

Algorithm 5, *move_J()*

This movement type can only plan for one pose, meaning the only way to use this function is by iterating through the list of the planned path. The same acceleration and deceleration between the individual poses applies for this algorithm.

Algorithm 5 *move_J(target_poses)*

```
1: for pose in target_poses do
2:     move_group.set_pose_target(pose, end_eff_link)
3:     move_group.go()
4:     move_group.stop()
5:     move_group.clear_pose_targets()
6: end for
```

Algorithm 6, *move_waypoints()*

This algorithm computes a trajectory for the entire list of poses, where the resulting trajectory has some freedom for smooth movement between them. To test the robot arm's ability to follow the trajectory, the velocity and acceleration scaling are set to 1.0.

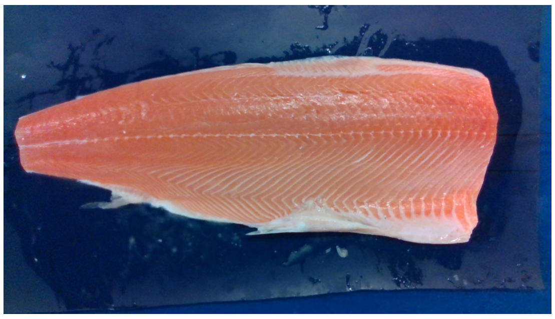
Algorithm 6 *move_waypoints(target_poses)*

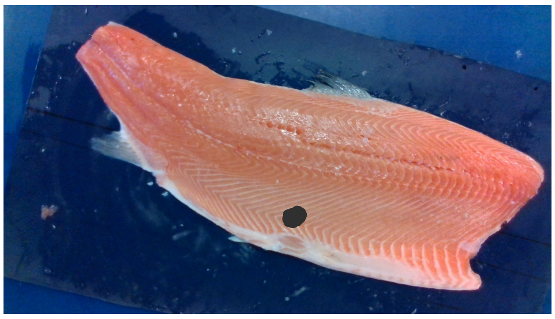
```
1: (plan, fraction) = move_group.compute_cartesian_path(target_poses)
2: plan = move_group.retime_trajectory(move_group.get_current_state(),plan,
   vel_scale=1.0, acc_scale=1.0)
3: move_group.execute(plan)
```

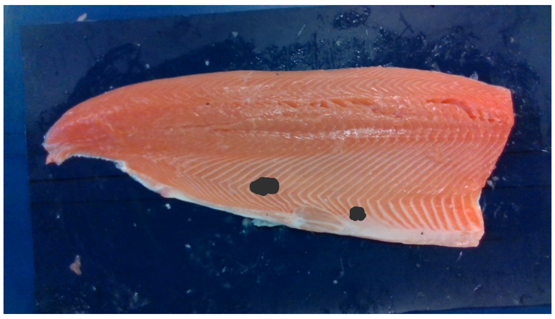
Chapter 4

Results and Discussion

This chapter will present and discuss the performance of the simulated robotic system, including the challenges that have arisen, limitations, and alternative approaches. The chapter is split into sections covering the steps of the built pipeline that lay the foundation for robotic scanning, before presenting and evaluating the performance of the simulated robot manipulation itself. The dataset consists of 87 images of fillets, where 29 of them have pseudo-melanin spots. Showing the results for each individual fish would be rather inexpedient. The results from the train:validation:test split meant that only one fillet in the test set has a pseudo-melanin spot. Therefore, some other samples were chosen and modified after the AI was trained to represent a bigger test set. This results in four different fillets, representing different numbers of melanin spots that will be used throughout this results section. The four fillets are shown in Figure 4.1.


(a) fillet_0


(b) fillet_1


(c) fillet_2

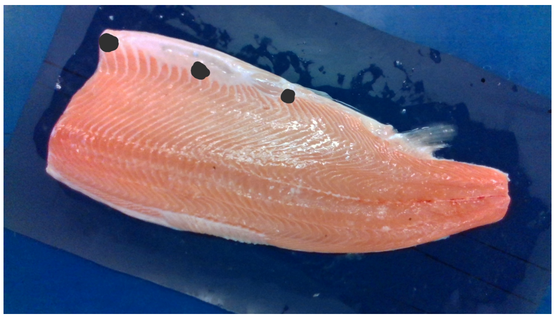

(d) fillet_3

Figure 4.1: Test fillets

The samples are named after the number of pseudo-melanin spots for simplicity, and they aim to test different cases for the AI, the planner, and the robotic scanning. It is worth mentioning that fish_3 is presented upside down, so the belly and the melanin spots are on the top side of the image. In addition to this, the fillet has the melanin spots at the far ends of the belly. This is done to test the system further.

4.1 AI and Feature Extraction

This section will go into more detail on the results of the AI, more specifically model training, some thoughts, and its performance as part of the robotic pipeline. Furthermore, the close related feature extraction will also be presented and discussed.

4.1.1 Model Training

The biggest limiting factor when it comes to the model training was evidently the use of the pseudo-melanin spots. The approach is not ideal to accurately mimic the real melanin spots, but as a proof of concept, it still demonstrates the potential of the system, and allows for easier testing of different melanin spot locations. Using a software tool to modify the limited number of pictures might also make it easier for the AI model to train and segment them, as there is a more defined border between the healthy part of the fillet and the melanin spots.

The loss function for the model, seen in Figure 4.2, shows that the system does indeed learn over the course of the iterations. However, at approximately 5000 iterations, it is apparent that the model starts overfitting, as it performs better on the training data than the validation data.

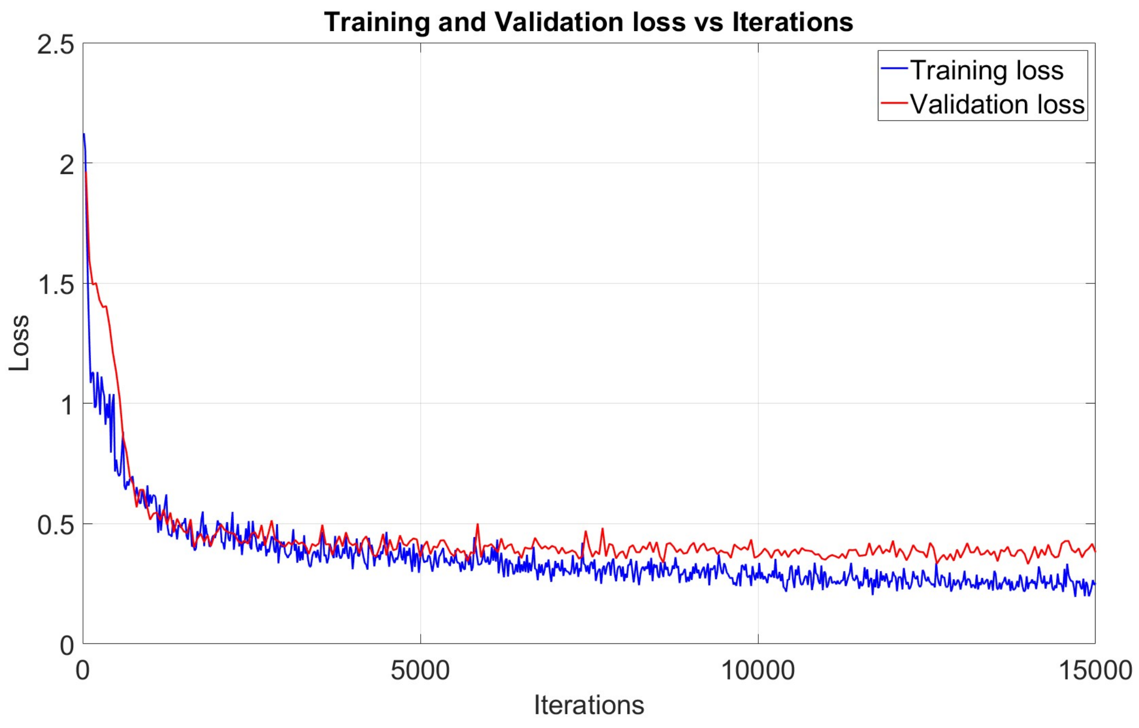


Figure 4.2: Training Loss

This observation suggests that 5000 iterations should be sufficient. A new model was therefore trained with early stopping at 5000 iterations. The loss function for this model is shown in Figure 4.3, and it is overall a decent result where the model learns throughout the iterations. However, it does show somewhat slow learning after 1000 iterations, as well as the first signs of overfitting towards the end of the training. The quality of the training data is limited both in terms of size and the use of pseudo-melanin spots. Therefore, the potential of a well-trained and generalized model is reduced, and this model was used for further experiments.

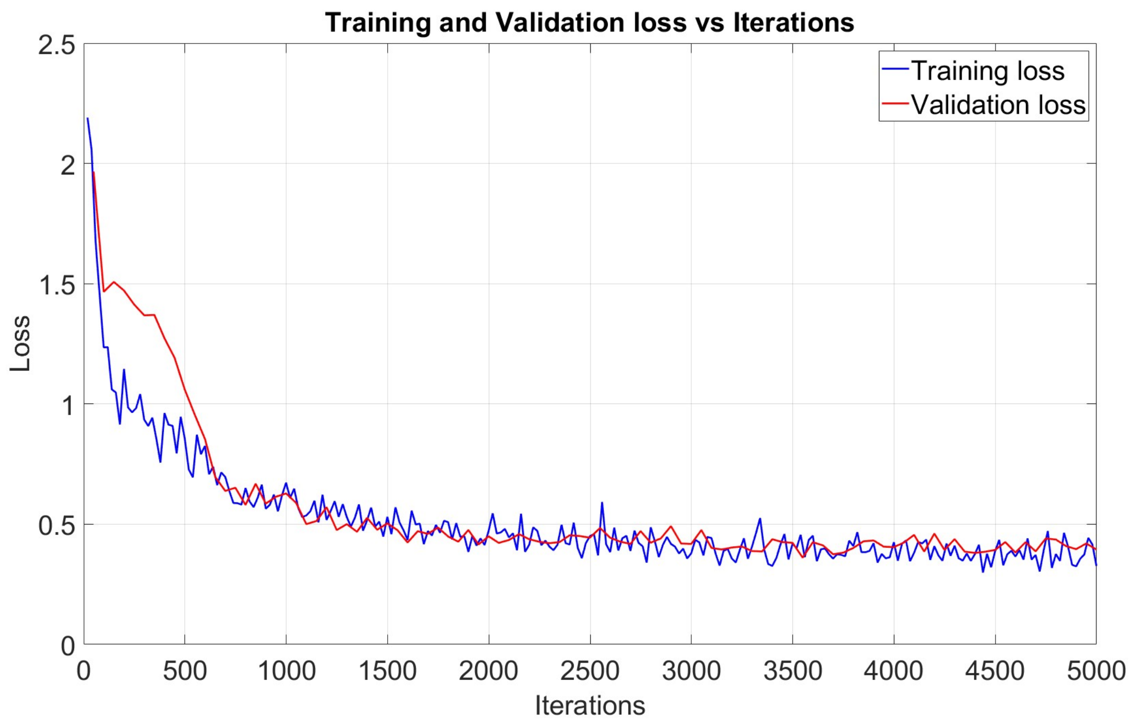
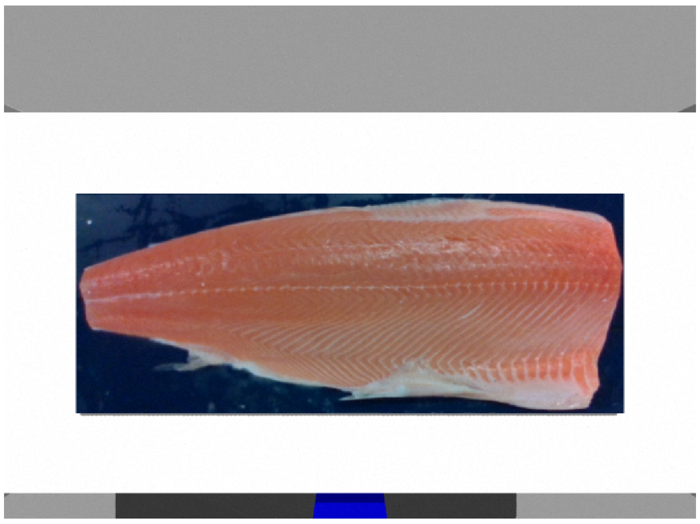


Figure 4.3: Training Loss

4.1.2 AI Segmentation Performance

To get a more precise evaluation of the AI performance, it was tested in the simulated environment on the unseen data. It was observed that both the segmentation of the belly, as well as the pseudo-melanin spot(s) for the most part were successfully detected with effectively no downtime in the continuous camera stream. This might be due to the limited data, but also that the pseudo-melanin spots are very distinct with little to no color gradient. However, the segmentation mask for the belly was, to a small degree, inconsistent toward the tail region of the fillet, where the height of the mask occasionally became narrower. This might be the result of poor image annotation of the dataset, lighting in the simulation, or other factors, like data quality.


(a) Realsense View

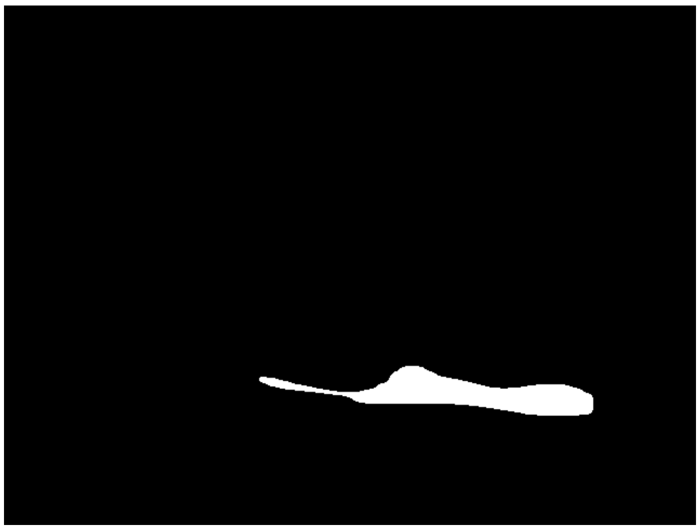
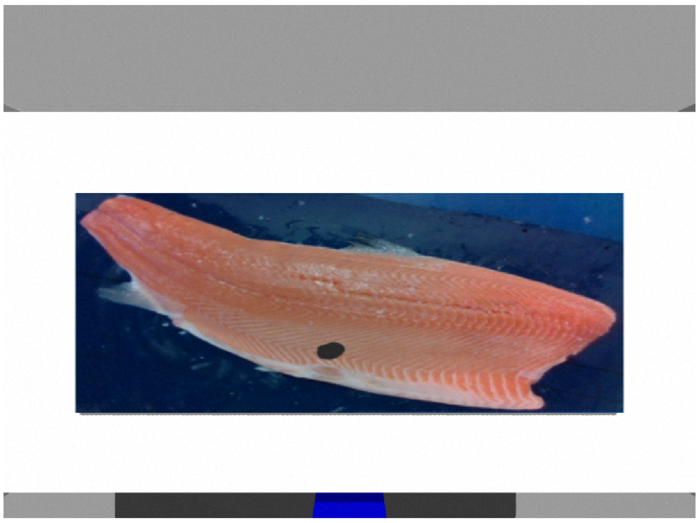
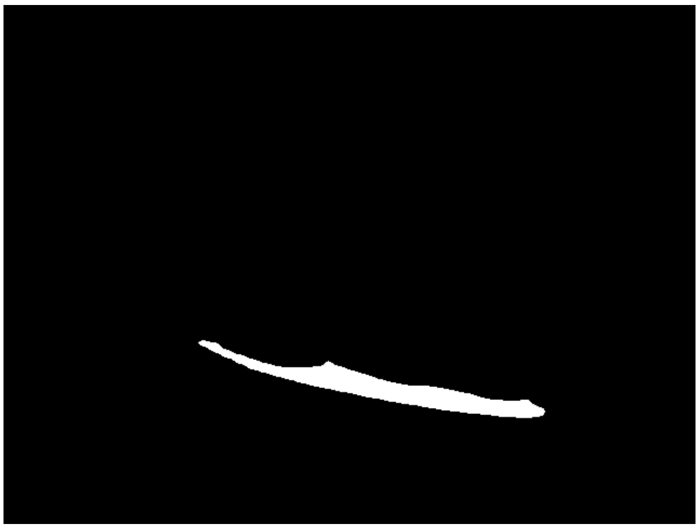

(b) Belly

Figure 4.4: AI Segmentation Masks for fillet_0.

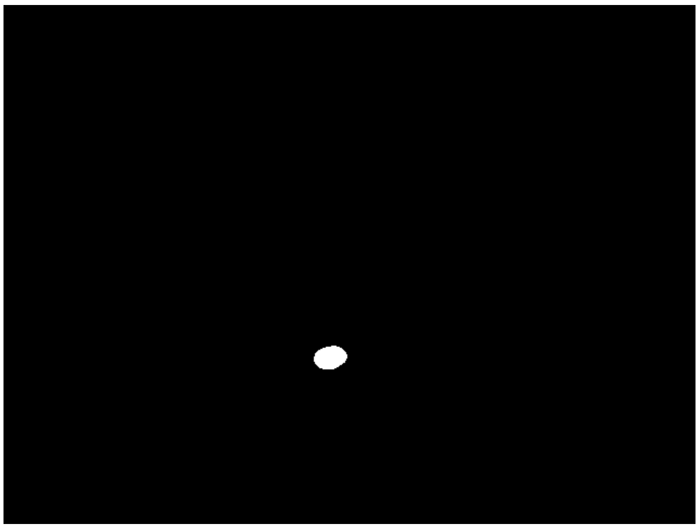
The result from the AI segmentation of fillet_0, seen in Figure 4.4, represents a simple case, where it does not have any melanin spots. However, the segmented belly (4.4b) is narrow toward the tail of the fillet, which only applied for this sample during the various tests.



(a) Realsense View



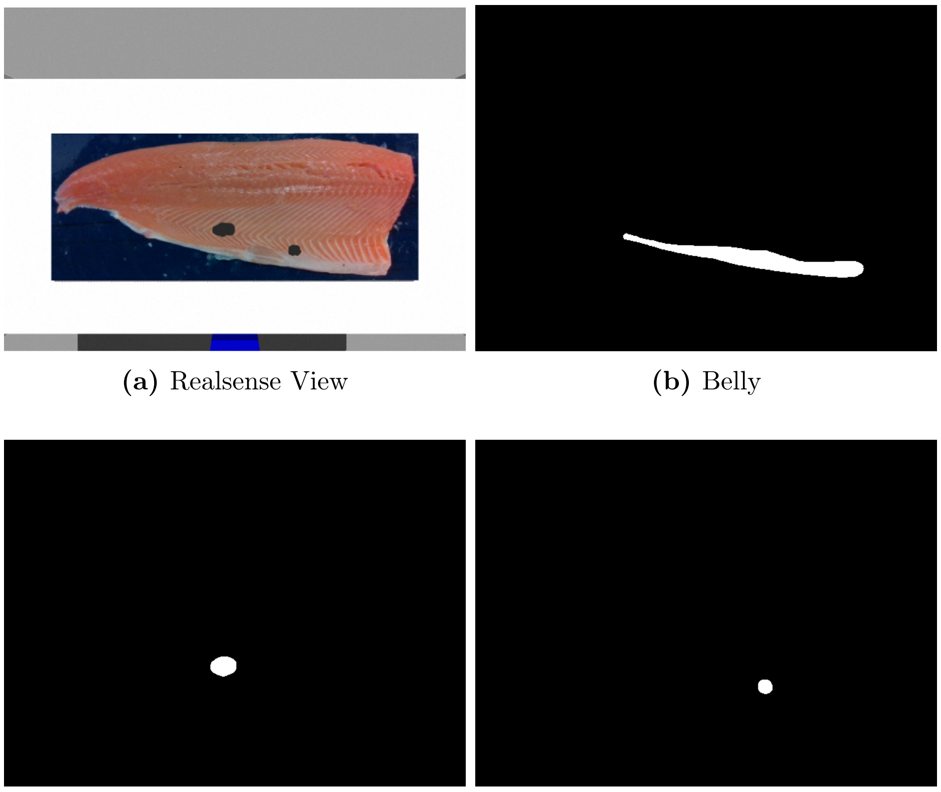
(b) Belly



(c) Melanin Spot

Figure 4.5: AI Segmentation for fillet_1.

fillet_1 aims to test another simple case for the AI where only one melanin spot is present. Figure 4.5 shows the two separate binary images returned by the AI. This fillet, contrary to fillet_0 has a more precise segmented belly, with no narrowing toward the tail region.


(a) Realsense View **(b)** Belly

(c) Melanin Spot **(d)** Melanin Spot

Figure 4.6: AI Segmentation for fillet_2.

The two melanin spots present on fillet_2 aims to represent a more difficult case. It showcases the systems instance segmentation, as the two spots are returned in separate binary images.

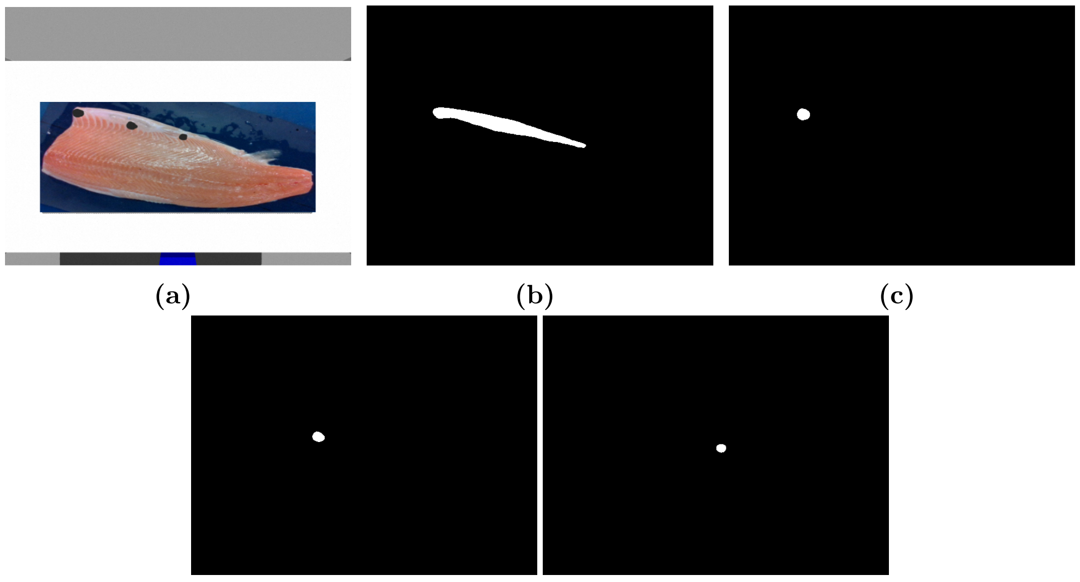

(a) (b) (c) (d) (e)

Figure 4.7: AI Segmentation fillet_3.

Even though fillet_3 is upside down, the AI manages to successfully segment the belly, as well as the three melanin spots, as seen in Figure 4.7. Since the training data is limited, the image augmentation provides the model with more samples to learn from, and the orientation for this fillet does not limit the segmentation.

Overall, the AI model performs quite well, even though it shows small signs of weakness in segmenting the belly, like for fillet_0. This encourages a better-trained model with more accurate annotations. It should be mentioned that instead of using instance segmentation to detect the belly, object detection should be considered, as this simpler approach might give better results, and it does not make much sense to use a complex model like instance segmentation when there is only one belly present.

As for the melanin spots, the AI showed good performance with no signs of real downtime. This might be because all the spots have the same color, and that they have no smooth transition to the healthy part of the fillet. It is hard to say how well this ad hoc solution will carry over to the real melanin spots, which might not be as well defined as the pseudo-melanin spots in this thesis. However, as a proof of concept approach, it shows promising results. The model demonstrates promising behaviour on the belly, which is a less distinct part of the fillet than the discolored melanin spots. Therefore, it is not believed the model will have significant difficulties training and detecting real melanin spots, given a larger dataset with real melanin spots.

4.1.3 Feature Extraction

The feature extraction algorithm is overall consistent with finding q_s and q_g from the binary mask. However, as mentioned in Section 4.1.2, the AI occasionally returns narrow or oddly shaped masks of the belly, see Figure 4.4a. With the implemented approach of fitting a line through the segmented mask of the belly, explained in Section 3.4.1, this could cause this line to not pass through this narrow part, meaning q_s would be misplaced. This could either encourage a better-trained AI-model, or a more robust approach for finding q_s and q_g, like choosing the top or bottom corners of a bounding box from object detection. However, the feature extraction seems to be consistent apart from this case.

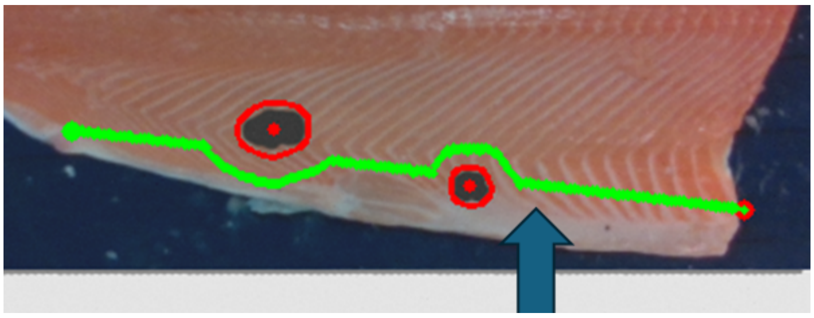
As for the melanin spots, it is hard to really evaluate the effect of using convex hull to avoid local minima problems, since the pseudo-melanin spots are not as big or complex shaped. The approach may also limit the free space the planner can use when circumventing a melanin spot. For the tested cases, however, it did not affect the planner in a negative or positive way. The approach of proportionally scaling an area $\mathcal{CO}$ by its original size might present problems for different sized, for example, considerably bigger or smaller, melanin spots, and it should be considered to rather scale by some constant margin.

4.2 Path Planning

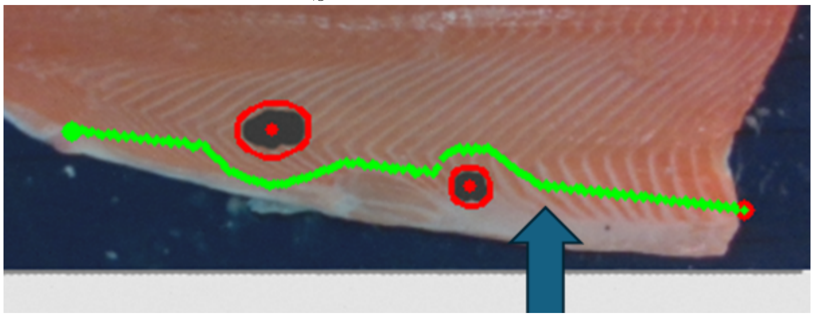
Before reviewing and discussing the general performance of the planner on the different test fillets, the parameters of the planner must be tuned, being $k_{a,goal}$ and $k_{a,desired}$. These parameters control the size of the steps toward q_g and the desired path respectively, and it will therefore affect plan quality and computation time.

4.2.1 Tuning the Planner Parameters

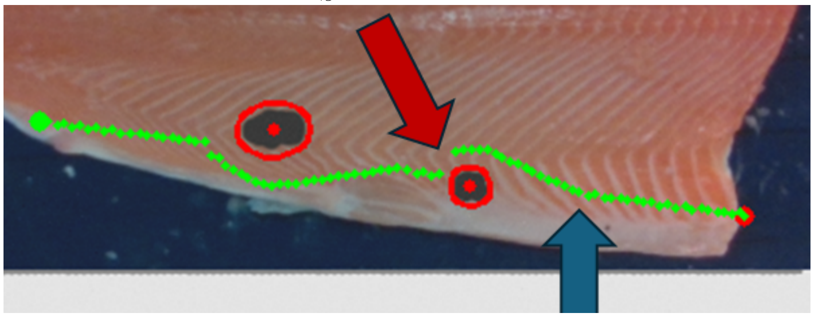
Choosing an appropriate value for the attractive force gain $k_{a,goal}$ is important to ensure good performance. The parameter is the only one responsible for the progression towards q_g, meaning it also controls the computation speed of the step-wise planner. An increased gain will therefore mean bigger, thus fewer steps, and faster computation time, at the cost of quality. The increased steps will make the planner become decreasingly accurate, as it is not exposed enough to the different fields for them to be effective, until some point where it can cause the planner to skip over a melanin spot. For this testing, fillet_2 was chosen, as it has a moderate number of melanin spots to test the planner, as well as $k_{a,desired} = 1$.



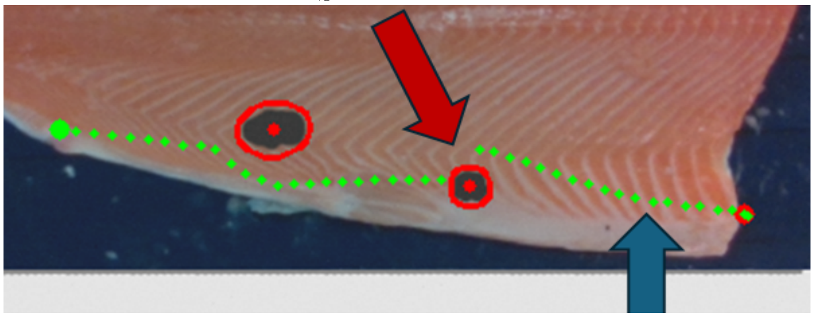
(a) $k_{a,goal} = 1$, time=110 ms



(b) $k_{a,goal} = 2$, time=37 ms



(c) $k_{a,goal} = 4$, time=32 ms



(d) $k_{a,goal} = 8$, time=7 ms

Figure 4.8: Different values for $k_{a,goal}$, RealSense View

Table 4.1: Attractive Force Tuning.

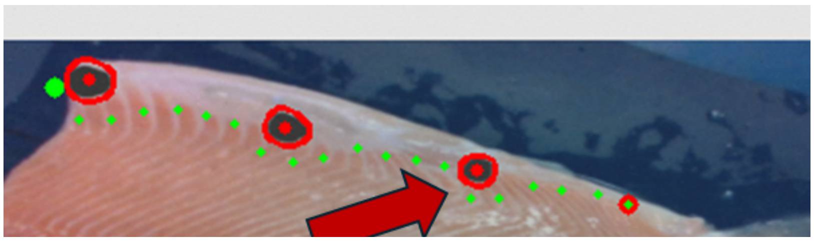
Attractive Force Tuning		
$k_{a,goal}$	Time	Continuous Plan
1	110 ms	Yes
1	130 ms	Yes
2	37 ms	Yes
2	50 ms	Yes
2	63 ms	Yes
4	27 ms	Yes
4	35 ms	Yes
4	32 ms	No
8	7 ms	No
8	15 ms	No

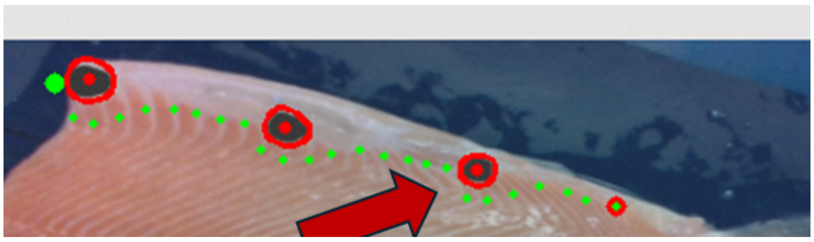
The results show varying degree of planner quality for the tested values of $k_{a,goal}$. It is also clear that this parameter has an impact on the planner's ability to follow the desired path, as highlighted by the blue arrows in Figure 4.8, which shows the view of the RealSense from its pose above the conveyor belt. An increased value of $k_{a,goal}$ results in fewer steps, resulting in less exposure to the force towards the desired path. Figure 4.8d shows the planned steps rejoining the desired line close to q_g, while Figure 4.8a shows the planner rejoining the desired path relatively quickly. The planned path gets notably worse at $k_{a,goal} = 4$, where it starts becoming discontinuous, marked by the red arrow in Figure 4.8c and 4.8d. It is also evident that the computation time is inversely proportional to $k_{a,goal}$, and even though it is relatively small for all the cases, it seems excessive to use the high detailed path generated by $k_{a,goal} = 1$. A good balance between computation time and performance seem to be $k_{a,goal} = 2$. With this value, the planner also shows promising ability to rejoin the desired path using $k_{a,desired} = 1$, which is why these values will be used for the presented results in Section 4.2.3.

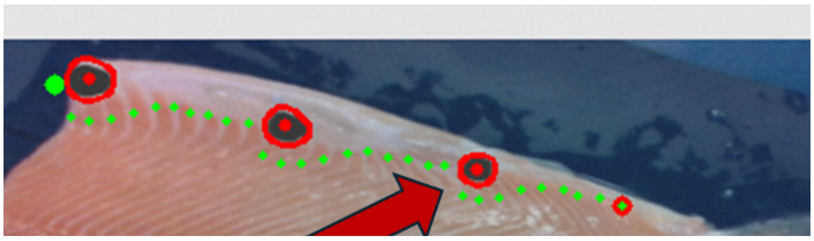
4.2.2 Choosing An Appropriate n_steps

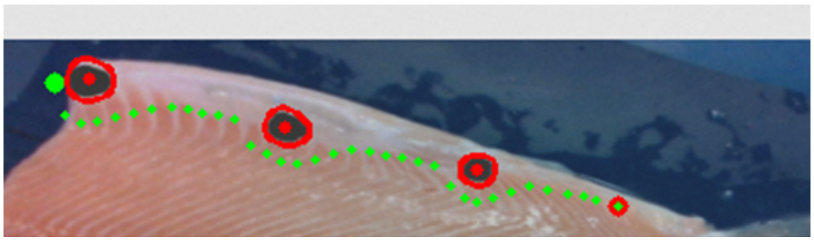
In contrast to the planner, which benefits from a high number of steps to get the full effect of the force fields, the waypoints set for the robot arm do not. Too many waypoints will force the arm to go through an unnecessarily detailed plan with little freedom for the trajectory between them, while too few will result in unsatisfactory melanin spot avoidance. It is therefore necessary to find an appropriate number of steps, n_steps for the robot arm.

Since the complexity of the planning space will have a significant impact on the required number of steps, for example, fillet_0 will effectively only require two steps, while fillet_3 will require many more, fillet_3 was chosen for testing different values.


(a) n_steps=20


(b) n_steps=25


(c) n_steps=30


(d) n_steps=35

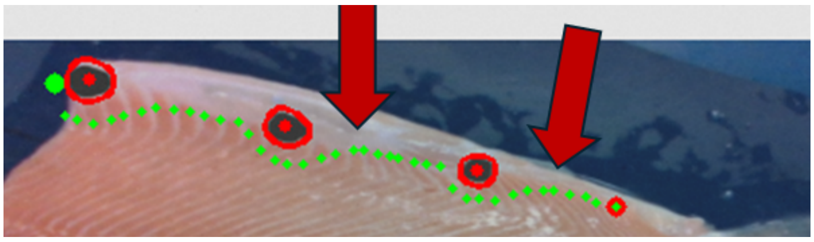

(e) n_steps=40

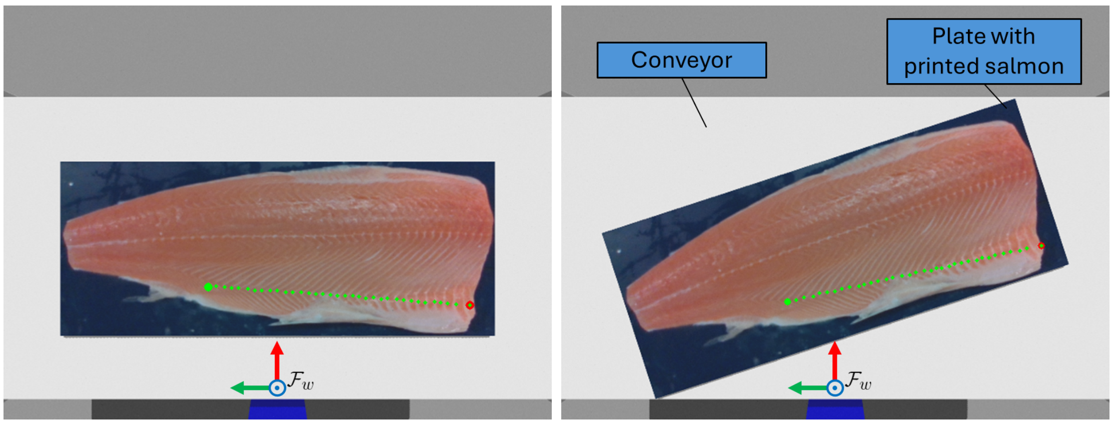
Figure 4.9: Different Values of n_steps

From the tested values, it is can be observed that $n\_steps \leq 30$ (Figure 4.9a-4.9c) show some tendencies to discontinuity, which might result in unsatisfactory melanin spot avoidance, while 40 steps (Figure 4.9e) results in close to overlapping waypoints. Setting $n\_steps = 35$ seem to offer a balance between them, the path looks quite continuous, while not having these dense, waypoints, as seen in Figure 4.9d. Based on these observations, 35 steps were chosen for further testing.

4.2.3 Planned Paths

The parameters used to obtain the results in this section are the following: $k_{a,goal} = 2$, $k_r = 1$, $k_{a,desired} = 1$, $goal\_thresh = 2$, $max\_iterations = 200$, $k_{scale} = 2$, and $n\_steps = 35$. The test samples are presented both relatively parallel with the conveyor and with an angle of $\psi = +18°$ about the z-axis. This angle was chosen as it is close to the maximum angle the plate with the printed salmon can be presented before interfering with the robot base, and hanging over the side of the conveyor.

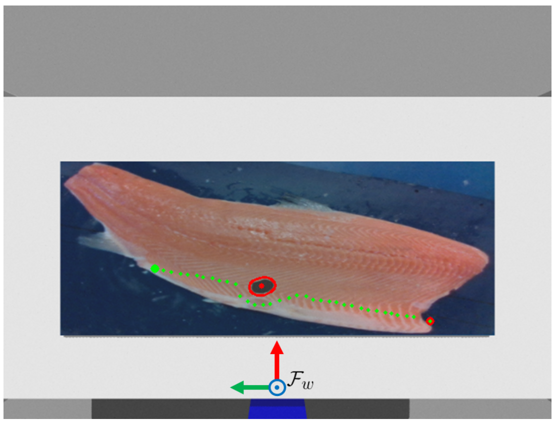
fillet_0



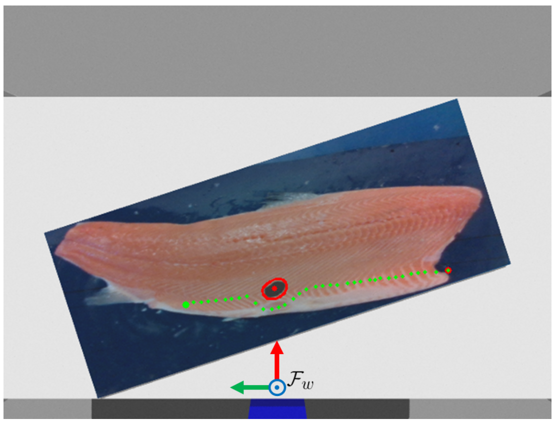
(a) Parallel Placement **(b)** Incline Placement

Figure 4.10: Planned Path fillet_0

fillet_1



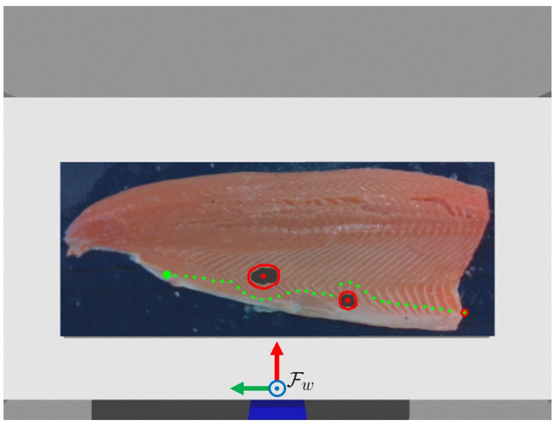
(a) Parallel Placement



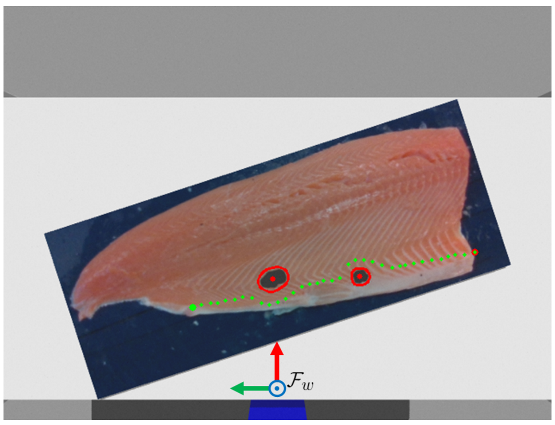
(b) Incline Placement

Figure 4.11: Planned Path fillet_1

fillet_2



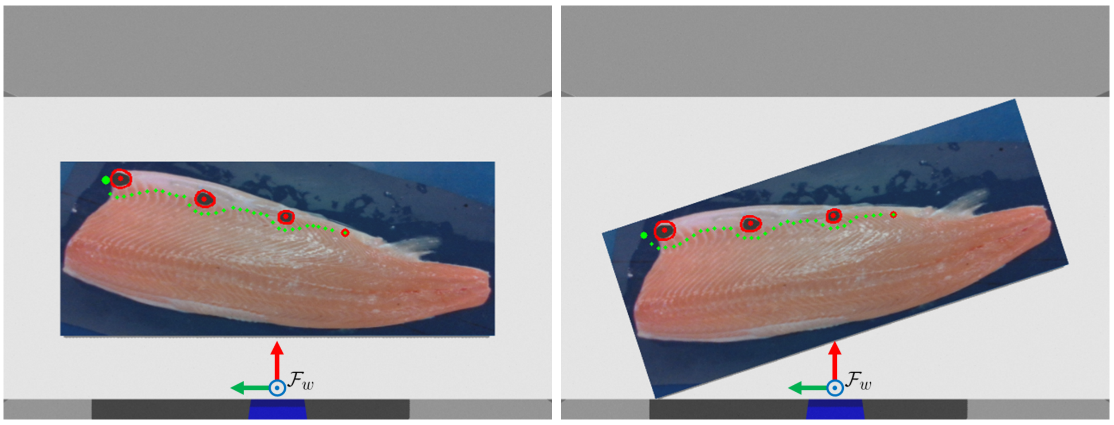
(a) Parallel Placement



(b) Incline Placement

Figure 4.12: Planned Path fillet_2

fillet_3



(a) Parallel Placement **(b)** Incline Placement

Figure 4.13: Planned Path fillet_3

Discussion

Starting with the simple fillet_0, the planner generates the desired straight line regardless of the orientation, as seen in Figure 4.10. It can be seen, that the straight line has some minor deviations. This might come from rounding of angle values throughout the planner calculations, and the effect of $f_{a,desired}(q)$. However, these deviations are not significant. As mentioned in Section 4.2.2, for this fillet, two points would be sufficient to describe the scan line. The 35 chosen steps has its disadvantages, as it generates a detailed path with these small deviations.

For fillet_1, since the mass center of the melanin spot is above the desired path, the planner circumvents under the melanin spot by the logic of the planner before going back to the desired straight path relatively quickly, until it reaches the goal.

fillet_2 shows an example starting like fillet_1, where the mass center of the melanin spot lies above the desired path, repelling the planner under it. The second melanin spot is the opposite case, as the mass center lies below, repelling the planner above, and ensuring the desired path is followed when possible. For this fillet, the planner risks generating a path that lies outside the fillet. This might be because the scaled contour is relatively large, and it shows a potential weakness in the planner.

Finally, fillet_3 demonstrates the planner's ability to consistently circumvent melanin spots and follow the desired path throughout the planner steps. It also starts close to a melanin spot, and is immediately repelled away from it, as seen in Figure 4.13

The choice of using 35 evenly spread waypoints shows varying degrees of importance. For fillet_0, two waypoints, namely q_s and q_g would be enough information for the arm, but as the complexity of the planner space $\mathcal{C}$ increases, more waypoints are needed. For the most complex planning scene, fillet_3, 35 waypoints seem quite adequate. This parameter might need more tuning, based on the number, size, and shape of melanin spots in the real industry. The available data for testing is still small, and further testing is necessary to better evaluate the planner in even more cases that might not have been accounted for in this thesis.

4.3 Scan Execution

In this section, the results of the executed scans are presented. First, the performance of the three different robot movement types, as explained in Section 3.6, are presented, followed by further testing of the best of the three.

Figure 4.14 shows the deprojected 2D path from the path planner in the image plane to the 3D path for the robot in the world frame.

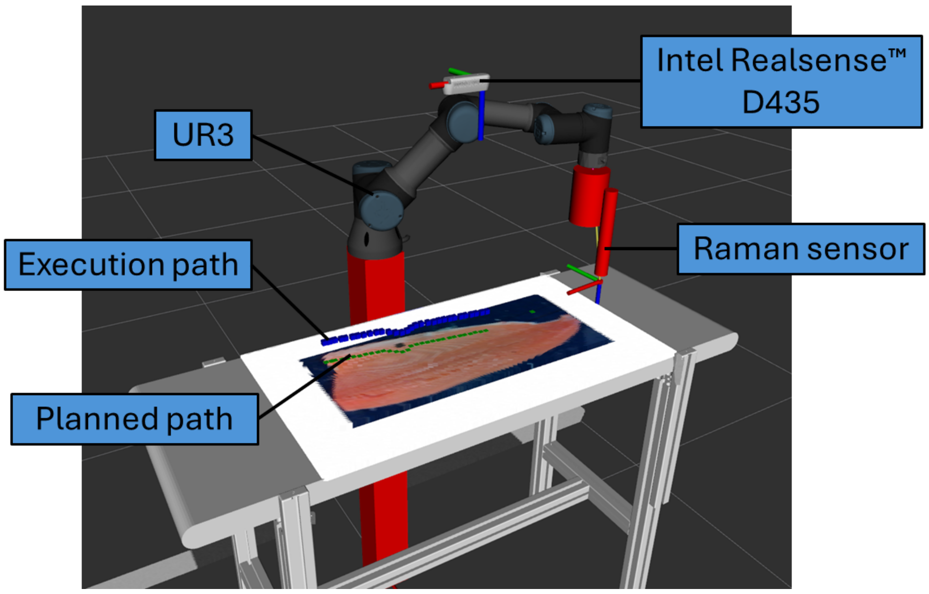
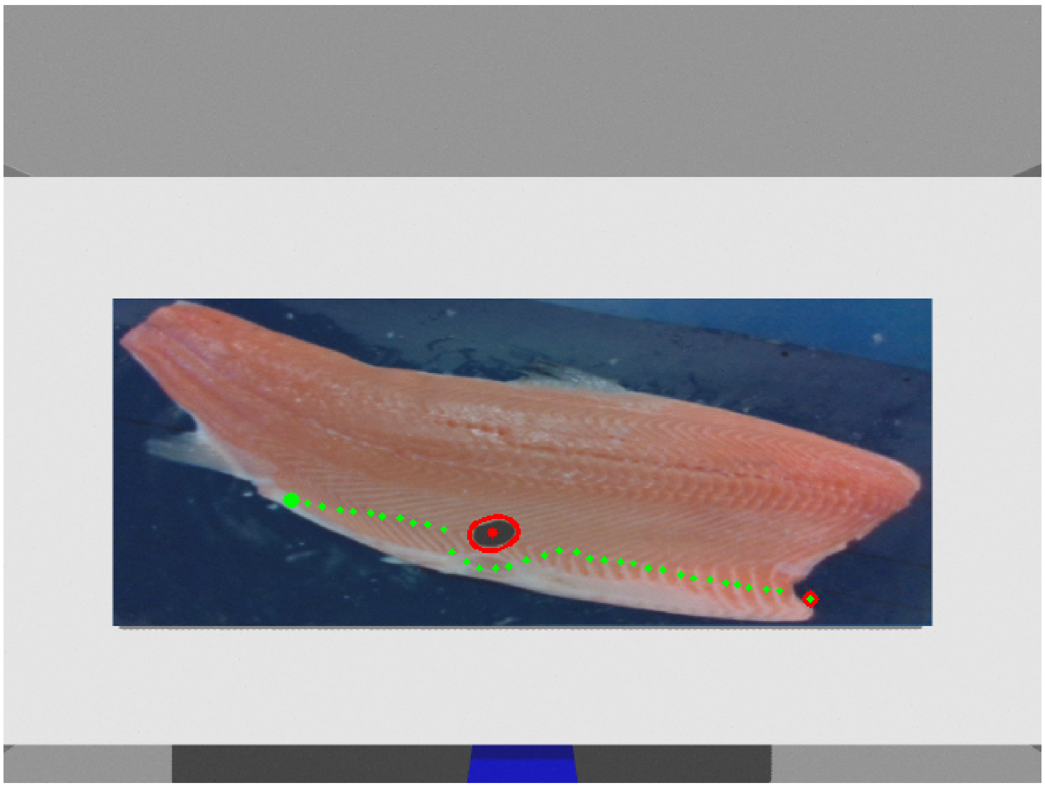


Figure 4.14: Planned and Execution Path.

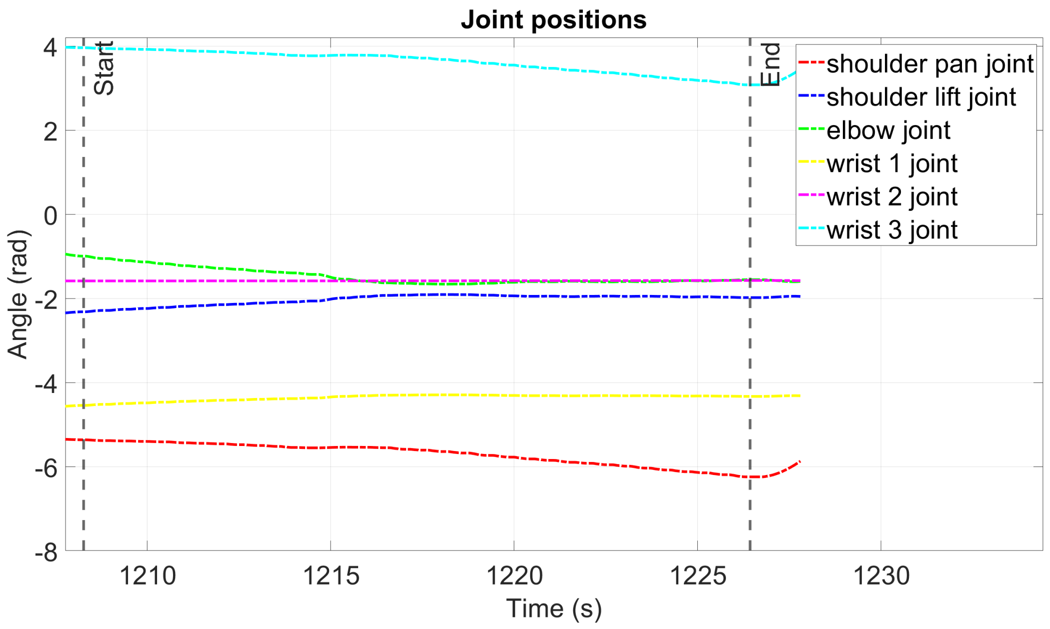
Movement Types

This part of the thesis will take a closer look at the performance of the three different robot movement methods discussed in Section 3.6. The following results show the joint angles plotted over time for the four different movement methods on the planned path of fillet_1, as shown in Figure 4.11a. The planned path is repeated below.

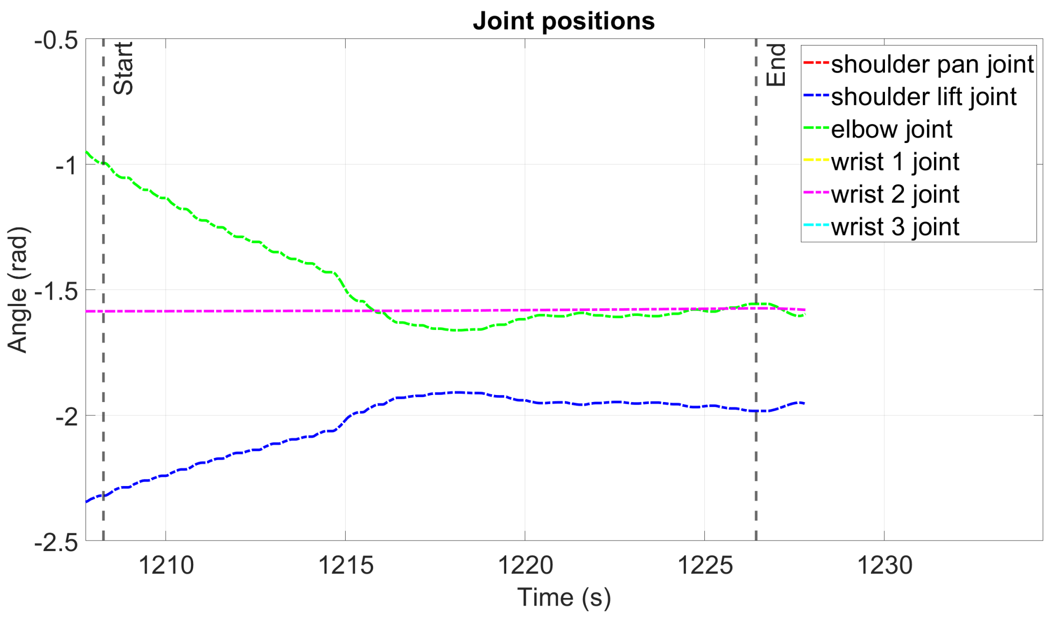


Planned Path for fillet_1

Alogrithm 5, *move_ J()*



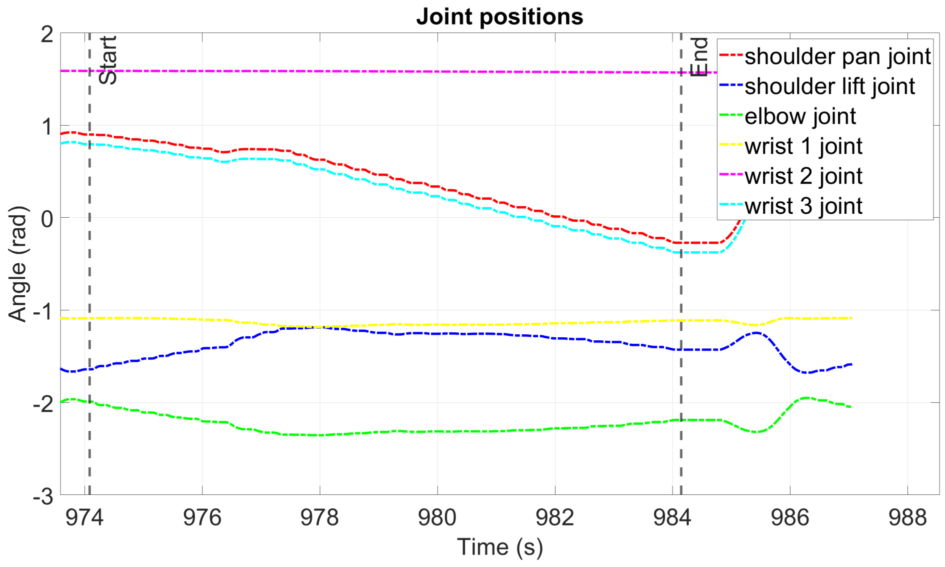
(a)



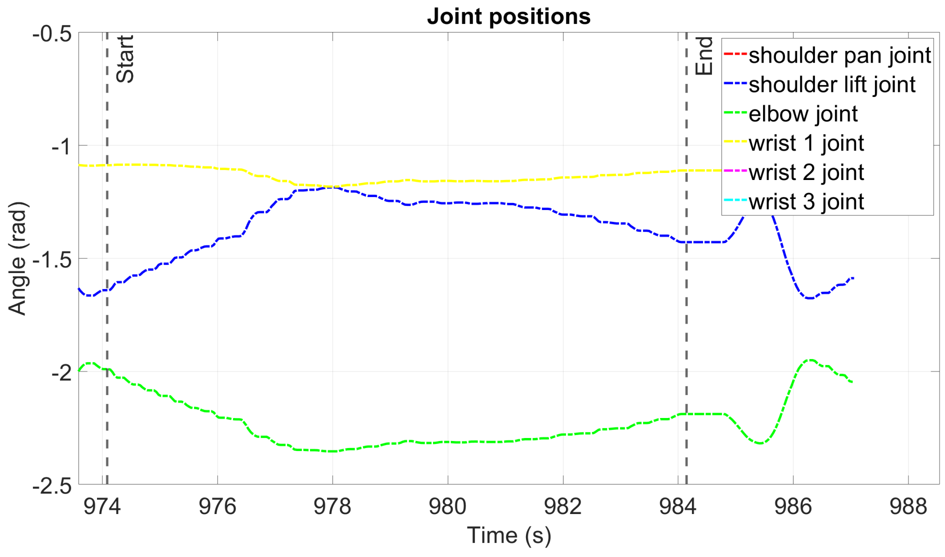
(b)

Figure 4.16: Plot demonstrating joint angles w.r.t time. Start of path at $t = 1208.269$, end of path at $t = 1226.429$.

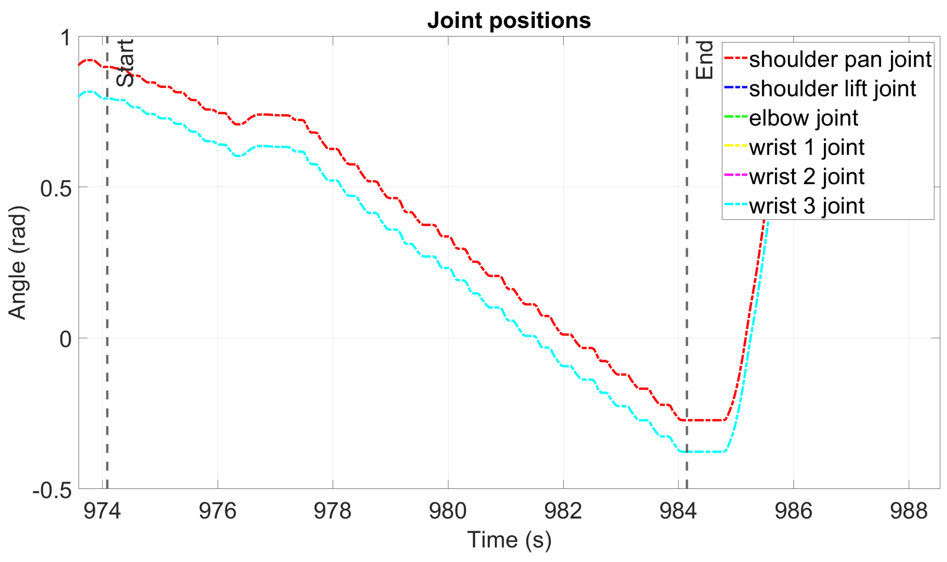
Algorithm 4a, *move_ L()*, (*vel_ scale* = 1.0, *acc_ scale* = 1.0)



(a)



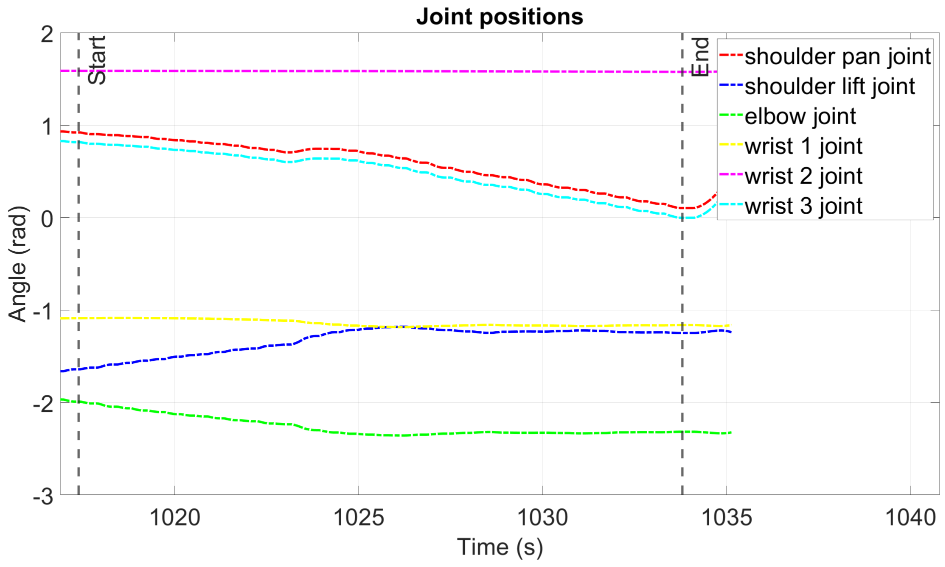
(b)



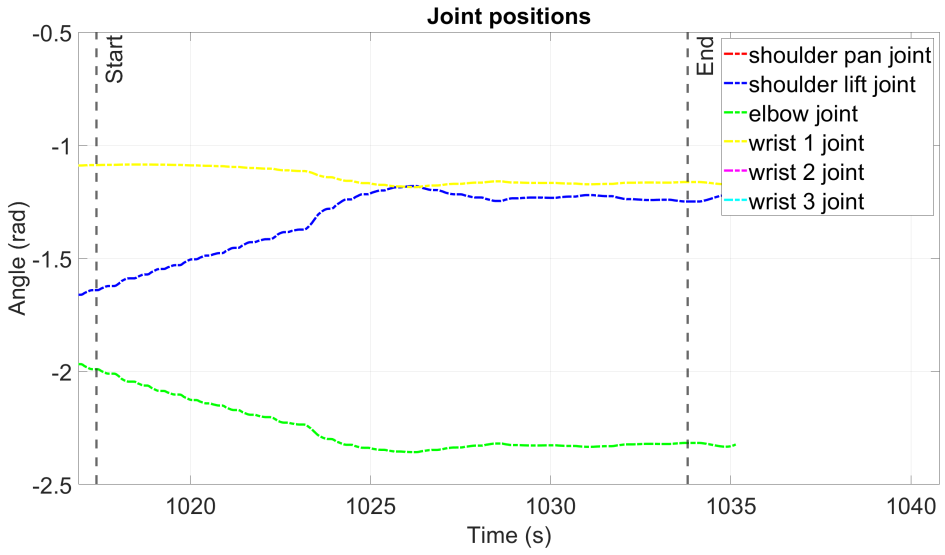
(c)

Figure 4.17: Plot demonstrating joint angles w.r.t time. Start of path at $t = 974.083$, end of path at $t = 984.143$.

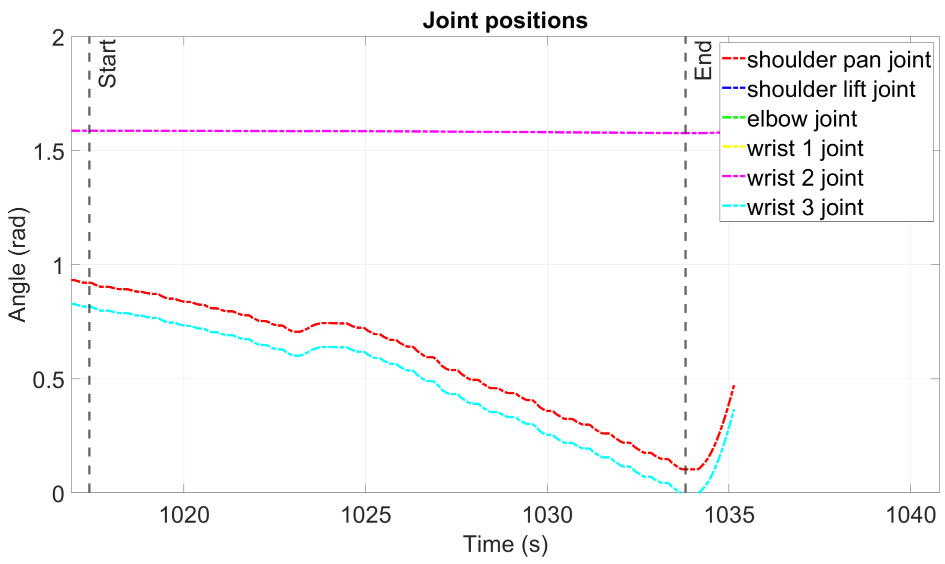
Alogrithm 4b, *move_L()*, (vel_scale = 0.2, acc_scale = 0.2)



(a)



(b)



(c)

Figure 4.18: Plot demonstrating joint angles w.r.t time. Start of path at $t = 1017.4$, end of path at $t = 1033.8$.

Alogrithm 6, *move_ waypoints()*

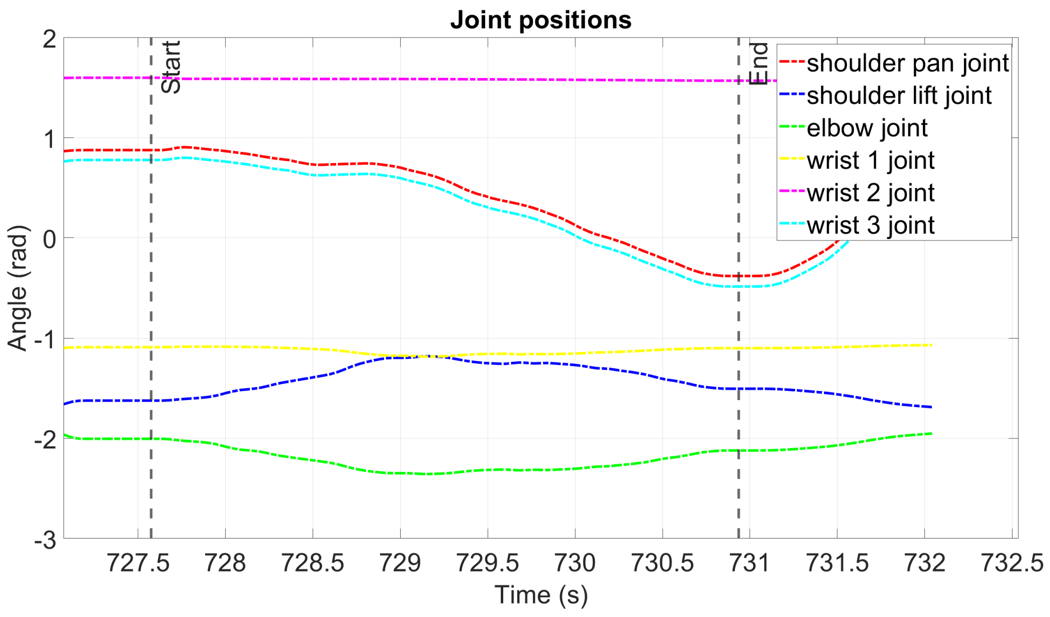


Figure 4.19: Plot demonstrating joint angles w.r.t time. Start of path at $t = 727.576$, end of path at $t = 730.936$.

Performance

The resulting execution times for the different methods are presented below:

Algorithm 5, *move_ J()* :	18.16 seconds
Algorithm 4b, *move_ L()* :	16.40 seconds
Algorithm 4a, *move_ L()* :	10.06 seconds
Algorithm 6, *move_ waypoints()* :	3.36 seconds

Algorithm 5, shown in Figure 4.16, shows that the iterative approach is inefficient. The execution takes 18.6 seconds, which is not viable for the potential of in-line measurements. Here, *wrist 3 joint*, also shows that the z-orientation of the dummy Raman sensor $\mathcal{F}_t$ is inverted. The step-wise motion throughout the execution becomes clearer in Figure 4.16b, which presents a more detailed look at the *elbow joint*, *shoulder lift joint* and *wrist 2 joint*. However, since the execution takes more than 18 seconds, this steps-wise motion is somewhat smoothed out.

The shortcomings of the iterative approach is also seen in Figure 4.17, and even though *move_ L()* is faster than *move_ J()*, it is still more than 7 seconds slower than *move_ waypoints()*. The acceleration and deceleration caused by the numerous individual trajectories become clear in Figure 4.17b and 4.17c. The results from the reduced velocity and acceleration scaling (Algorithm 4b), are seen in Figure 4.18. Here, it is easier to see that the arm gradually speeds up, as the joint angles are less jagged, which results in a slow execution time of 16.40 seconds.

Figure 4.19 shows the performance of Algorithm 6, when the trajectory is computed and executed based on the entire list of poses. The scan takes 3.36 seconds, which is approximately 3x times as fast as the iterative counterpart, Alorithm 4. The joint angles show a stable, smooth execution, which was also observed in Rviz at the time of execution.

It becomes clear that the Algorithm 6, *move_ list()*, performs the best. The scan takes 3.36 seconds, which is fast compared to 18.16, 16.40, and 10.06 seconds for the three tested iterative approaches. The iterative methods result in a slow execution time because they do not compute a single trajectory through all the waypoints, but rather calculate and execute individual trajectories between each waypoint of the list, making the arm accelerate and decelerate between each of them. This means the number of steps will have a great effect on the execution time for the iterative methods. However, reducing the number of steps will likely not make the iterative methods comparable with the non-iterative one, due to the acceleration between the poses. By running these tests in the simulation, it is possible to find issues and problems, which reduces the risks of testing the planner on a real arm.

4.3.1 TCP Tracking and Joint Results

As the non-iterative *compute_cartesian_path()* + *execute()* showed superior behaviour, its performance was further tested on the four different fillets: filet_0, filet_1, filet_2, and filet_3. The fillets are presented with the orientation of $\psi = 18°$ about the z-axis, as this is considered a greater challenge for the arm.

filet_0

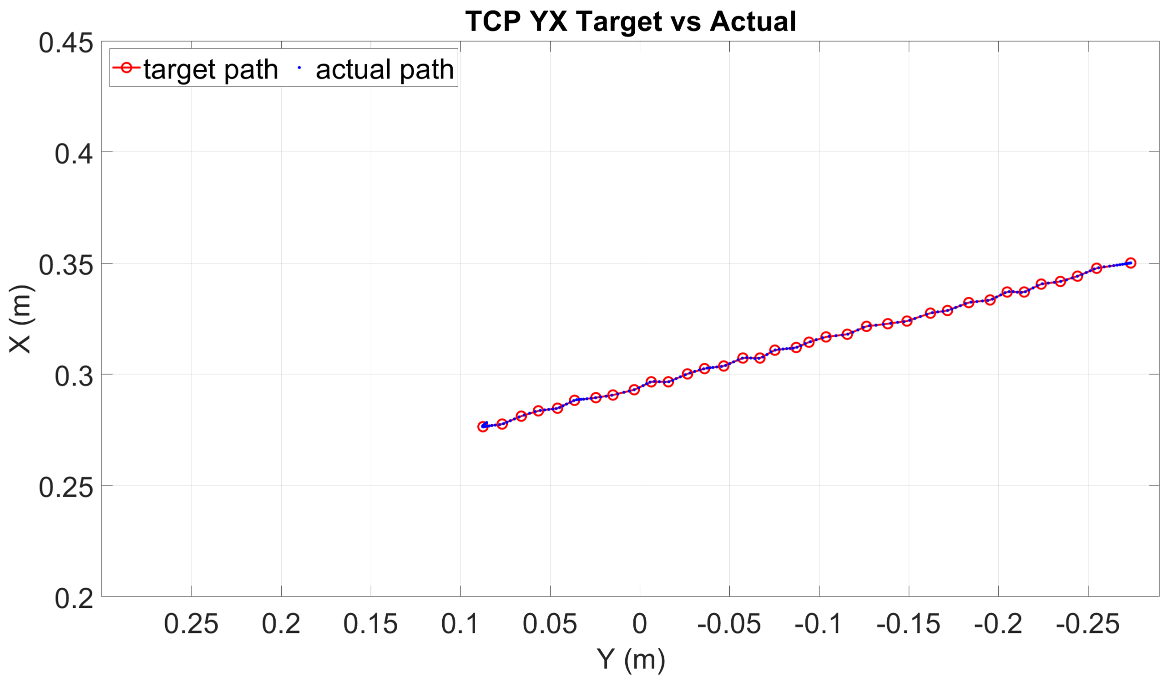


Figure 4.20: Executed Path Incline fillet_0

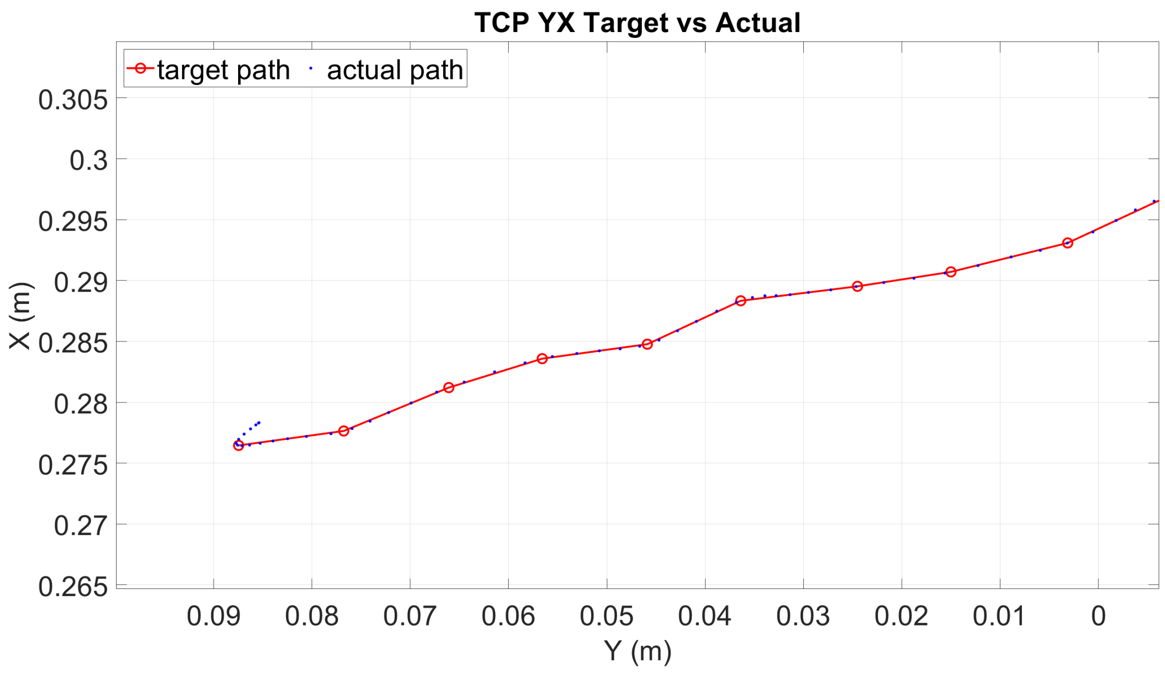


Figure 4.21: TCP Zoomed fillet_0

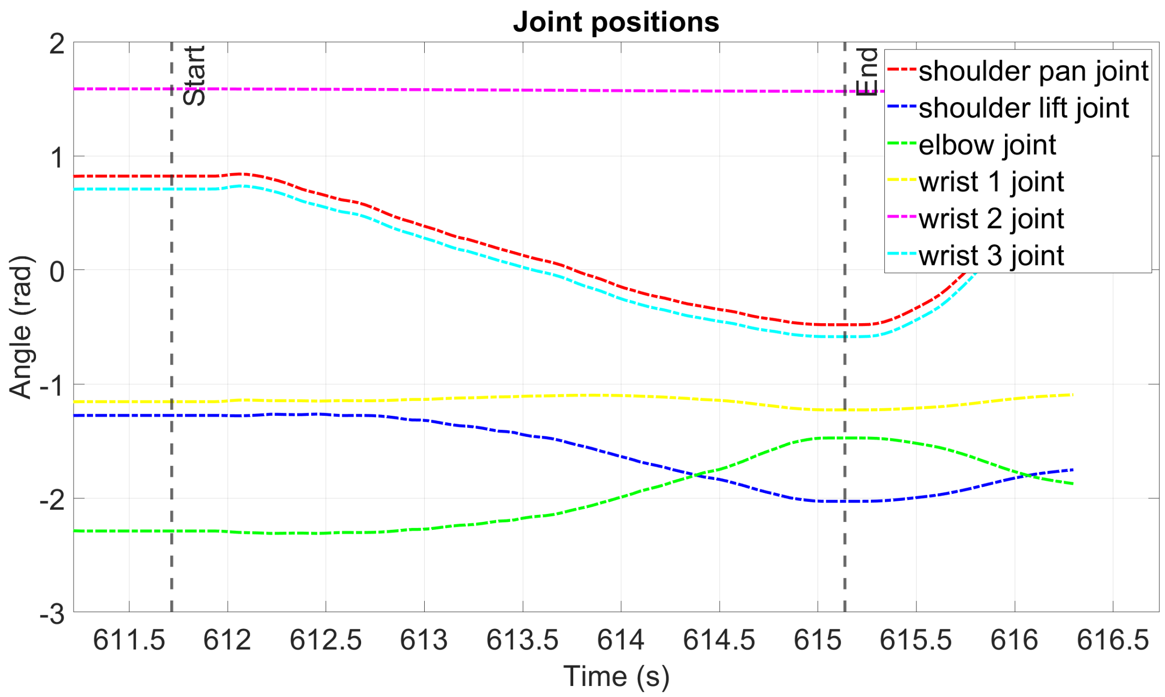


Figure 4.22: Plot demonstrating joint angles w.r.t time. Start of path at $t = 611.716$, end of path at $t = 615.136$.

filet_1

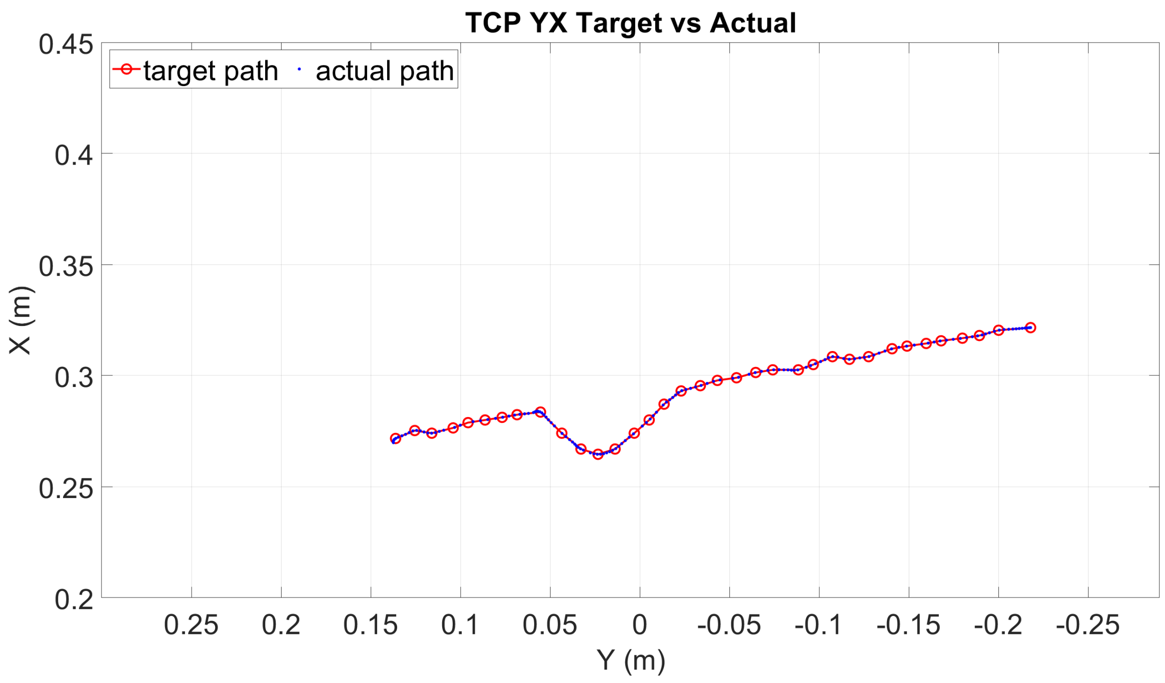


Figure 4.23: Executed Path Incline fillet_1

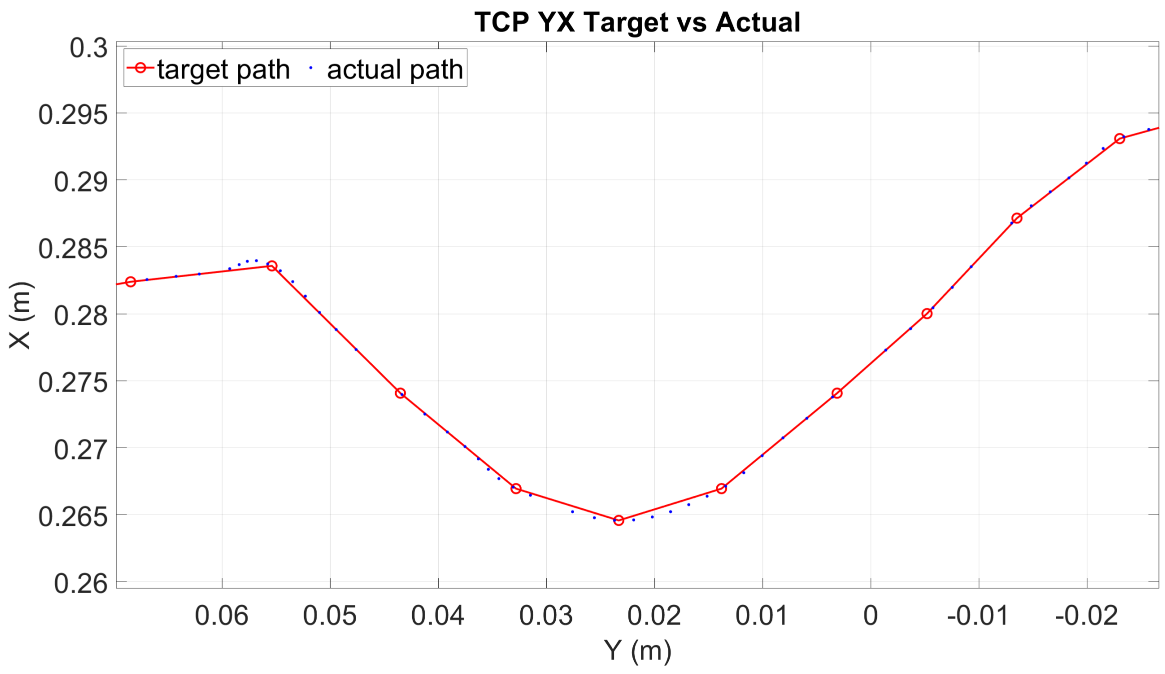


Figure 4.23: TCP Zoomed fillet_1

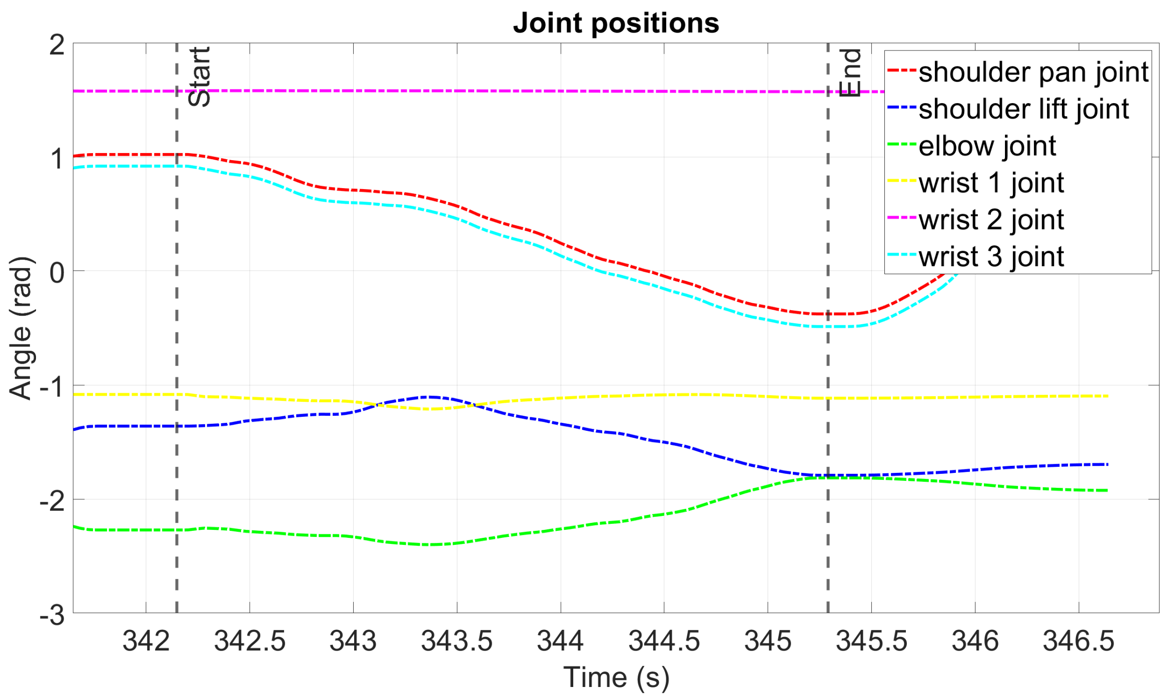


Figure 4.24: Plot demonstrating joint angles w.r.t time. Start of the path at $t = 342.149$, end of the path at $t = 345.289$.

filet_2

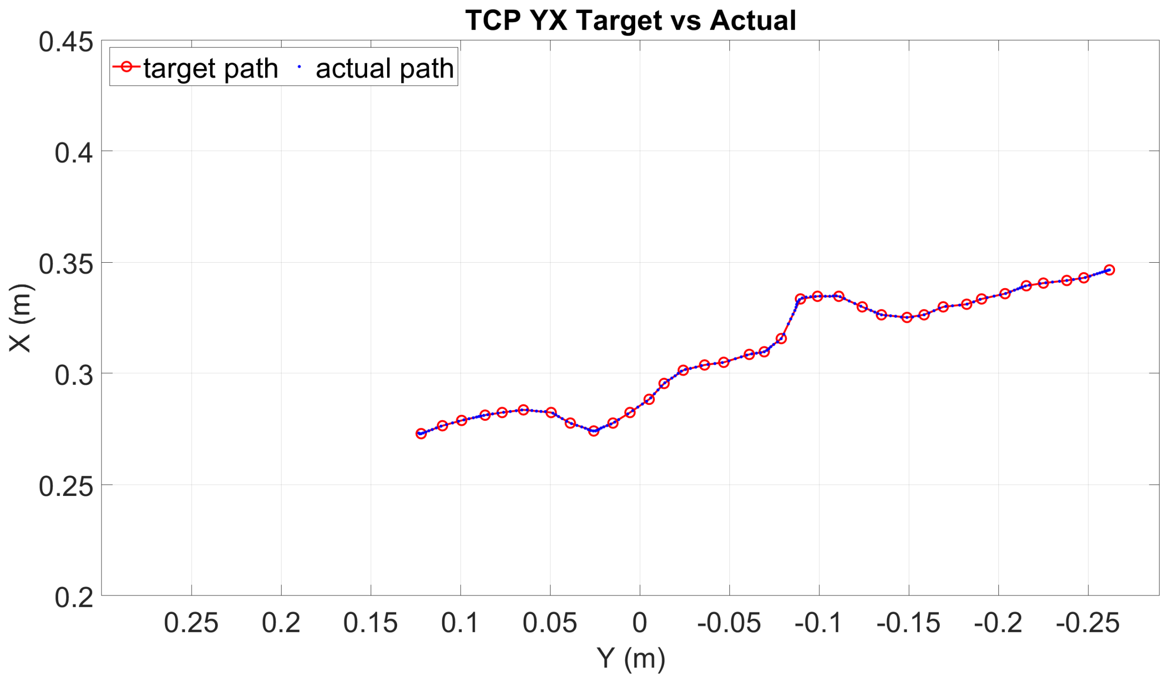


Figure 4.25: TCP Zoomed fillet_2

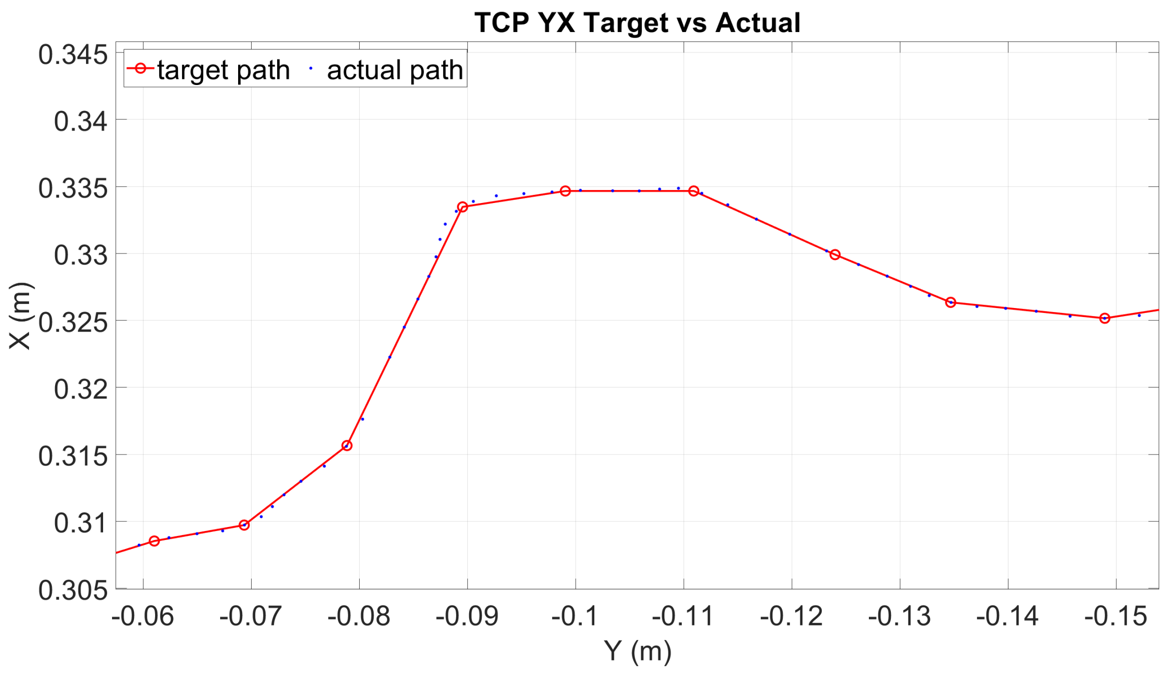


Figure 4.26: Executed Path Incline fillet_2

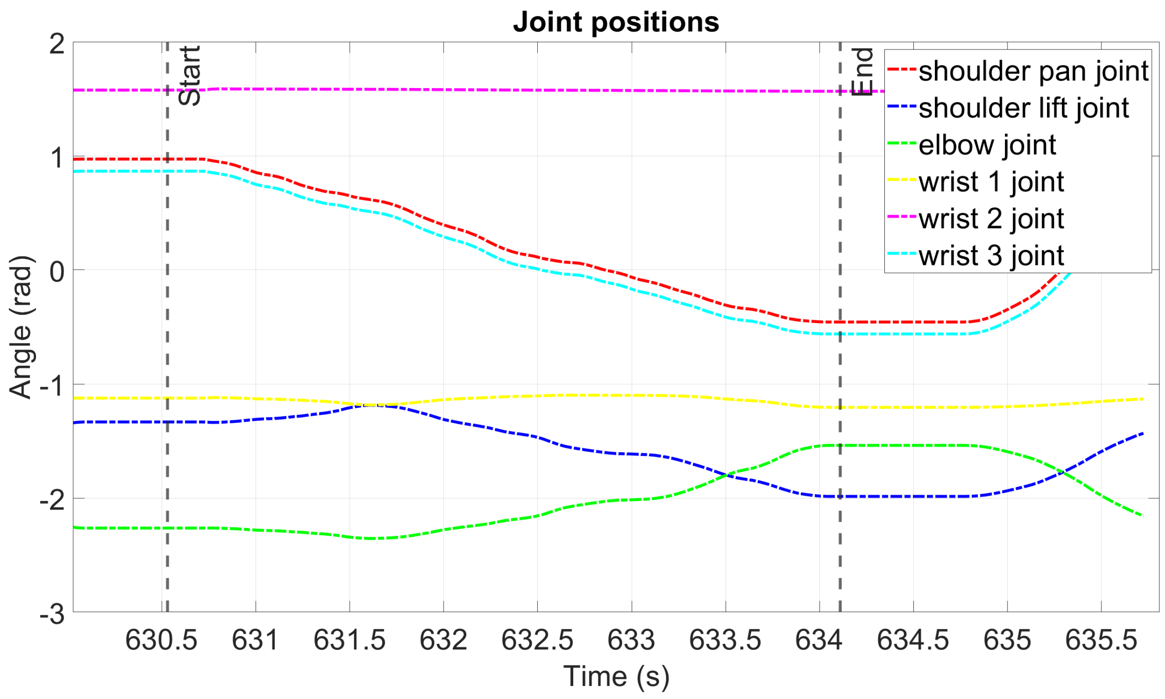


Figure 4.27: Plot demonstrating joint angles w.r.t time. Start of the path at $t = 630.529$, end of the path at $t = 634.109$.

filet_3

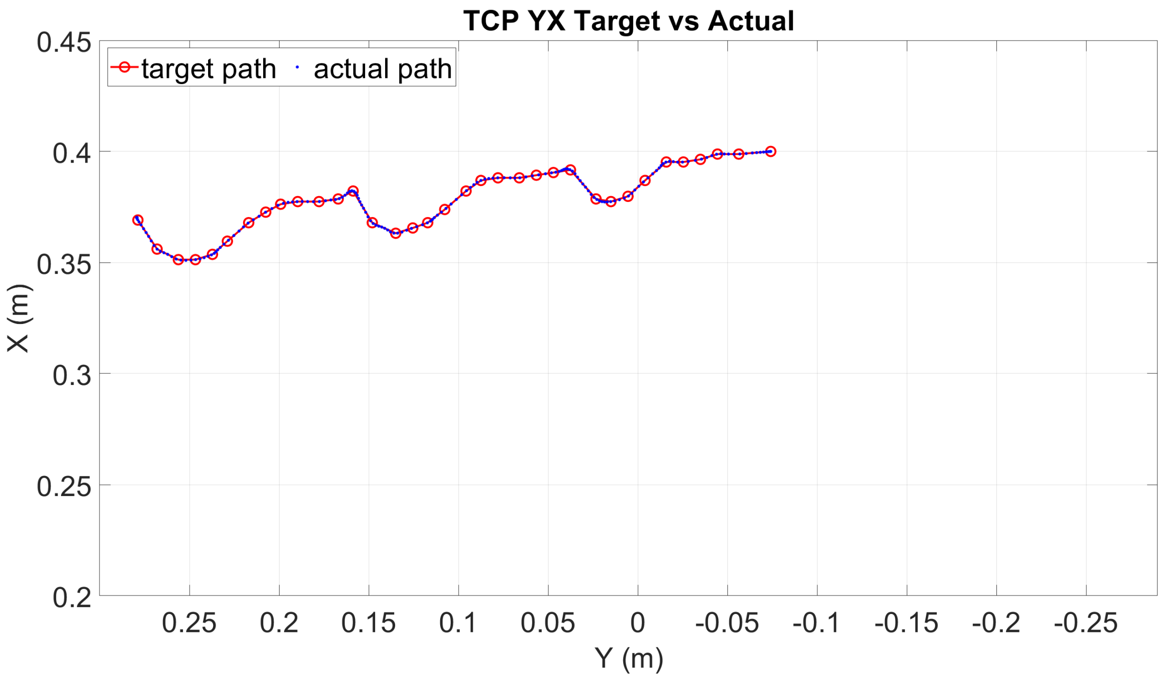


Figure 4.28: Executed Path Incline fillet_3

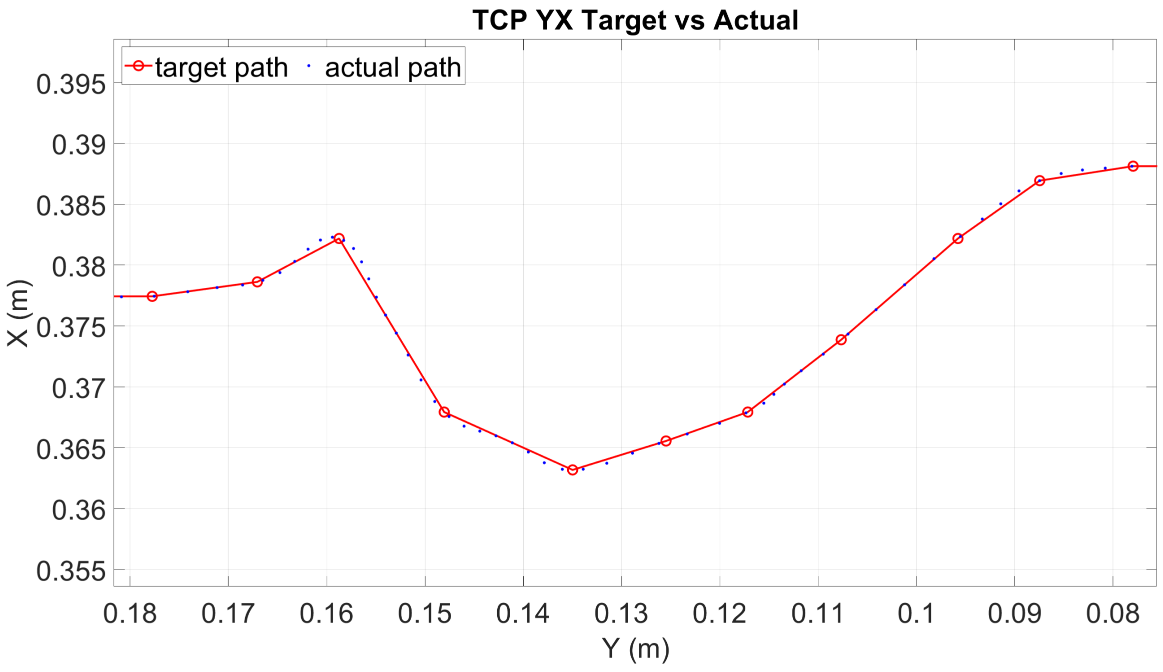


Figure 4.29: TCP Zoomed, fillet_3

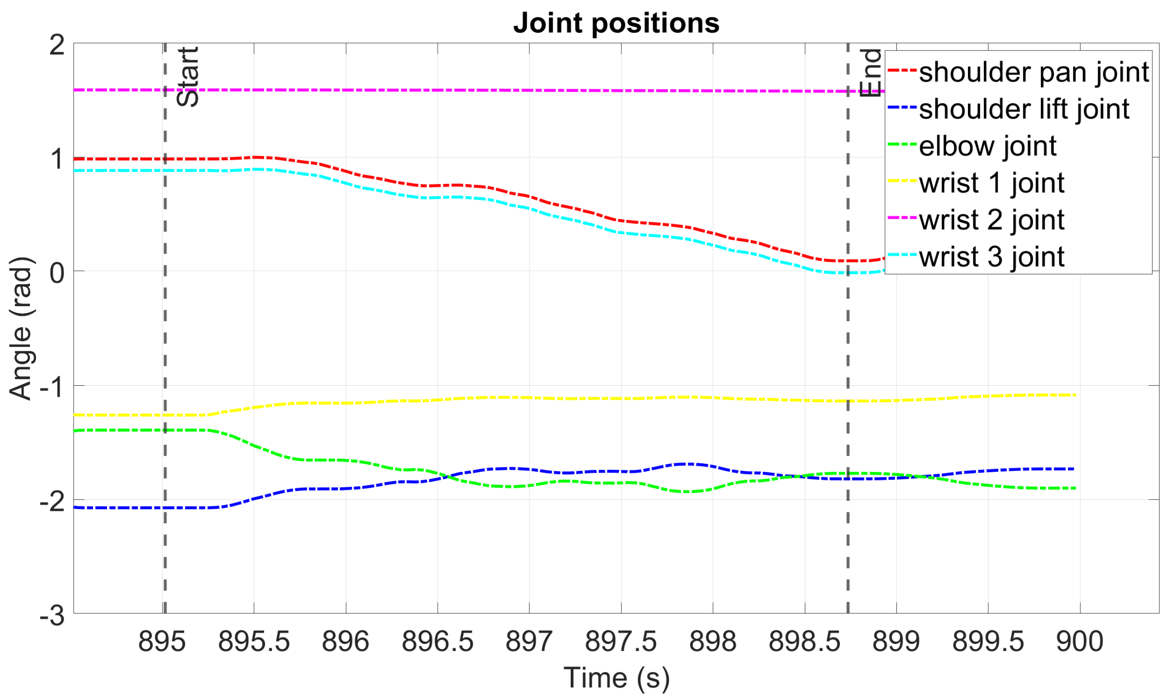


Figure 4.30: Plot demonstrating joint angles w.r.t time. Start of the path at $t = 895.015$, end of the path at $t = 898.735$.

Performance

The overall performance of the simulated scans seems promising. The plots in Figure 4.20 - 4.30 show the arm's ability accurately to track the waypoints for the TCP, as generated by the path planner, as well as the joint angles throughout the execution. The simple straight-line trajectory is presented by fillet_0. The TCP accurately goes through each of the waypoints with high accuracy, as seen in 4.20. A more detailed inspection of this trajectory is presented in Figure 4.21, and it can be observed that the actual path has a little 'tail' at the beginning $y \approx 0.09, x \approx 0.28$. This probably comes from some minor logging flaw, where the actual position started logging before it had reached the first waypoint. The joint angles are also quite smooth throughout the execution.

fillet_1, filet_2, and fillet_3 present different numbers of melanin spots and increasing complexity of the trajectories, which are handled well by the arm. The executed path is able to accurately go through all of the waypoints, however in these cases, it becomes clear that the arm does not move in a straight path between the waypoints, but instead creates a smoother and continuous trajectory. This is naturally more visible in the sharp turns, like fillet_1 ($y \approx 0.06, x \approx 0.285$), fillet_2 ($y \approx 0.09, x \approx 0.335$), and fillet_3 ($y \approx 0.16, x \approx 0.38$). This type of trajectory ensures smoother execution, and it will naturally reduce the wear on the joints in the arm. The execution for fillet_1 (Figure 4.23) also shows the same 'tail' as fillet_0, which again, is likely caused by a logging flaw.

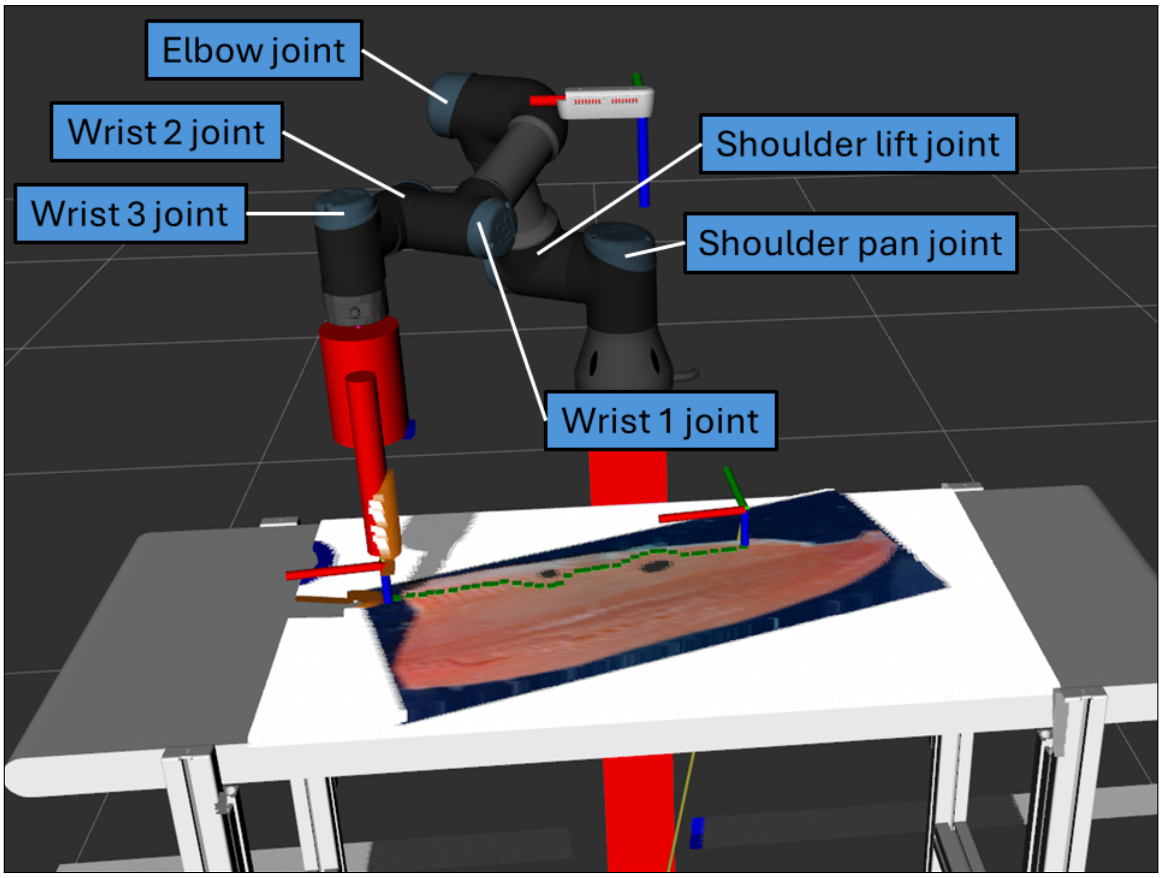


Figure 4.31: Joints of UR3 Arm

The joint angles for the four fillets show some general tendencies. To begin with, *wrist 2 joint* is stationary throughout the executions. This joint will change the orientation of the TCP in the YZ-plane of the world frame $\mathcal{F}_w$, which is not desired, since the TCP should be orthogonal to the table. It can also be seen that *shoulder pan joint* and *wrist 3 joint* follow each other throughout all of the executions, which ensures the orientation of the TCP about its own Z axis does not change. The *shoulder lift joint* and *elbow joint* seem to be countering each other, which makes sense, as the lift joint will raise the TCP in the world frame, and the elbow joint will counter this, ensuring the correct height of the TCP above the conveyor. The two (*shoulder lift joint* and *elbow joint*), together with *wrist 1 joint*, allow the arm to reach further out in the XY-plane of the world frame $\mathcal{F}_w$, with *wrist 1 joint* ensuring last corrections in TCP orientation.

Overall, the joint angles for the four fillets points to the complex 6-DoF arm being excessive for the task, and that a more simple robot manipulator might be sufficient to perform this robotic scanning, like a one- or two axis. However, if the task change slightly, like requiring more complex measurements or assessment, where the TCP must be oriented in a more complex way, the 6-DoF arm does prove its effectiveness.

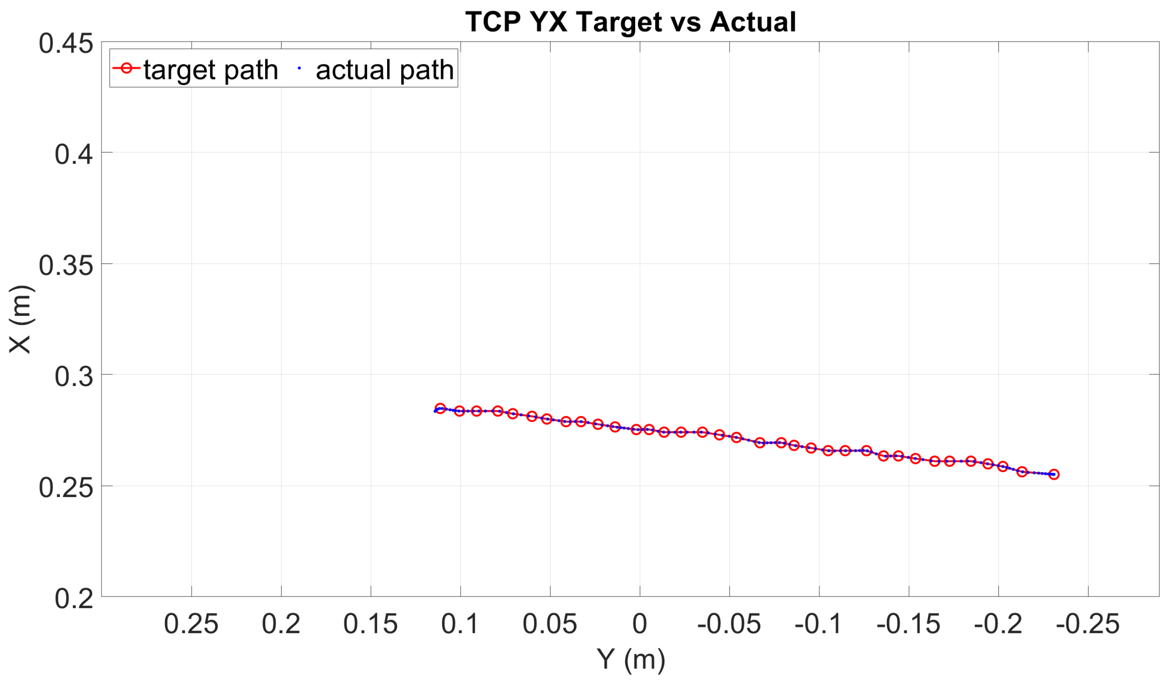
Table 4.2: Execution Times for Test Samples

Execution Times					
Sample	$z : \psi = 18°$	$z : \psi = 0°$	Mean	Scan Length	Avg. TCP Speed
fillet_0	3.42 s	2.56 s	2.99 s	0.3746 m	125.3 $mm\ s^{-1}$
fillet_1	3.14 s	3.36 s	3.25 s	0.3761 m	115.7 $mm\ s^{-1}$
fillet_2	3.58 s	3.94 s	3.76 s	0.4124 m	109.7 $mm\ s^{-1}$
fillet_3	3.72 s	3.20 s	3.46 s	0.3961 m	114.5 $mm\ s^{-1}$
Mean	3.465s	3.265 s	3.365 s	0.3898 m	116.3 $mm\ s^{-1}$

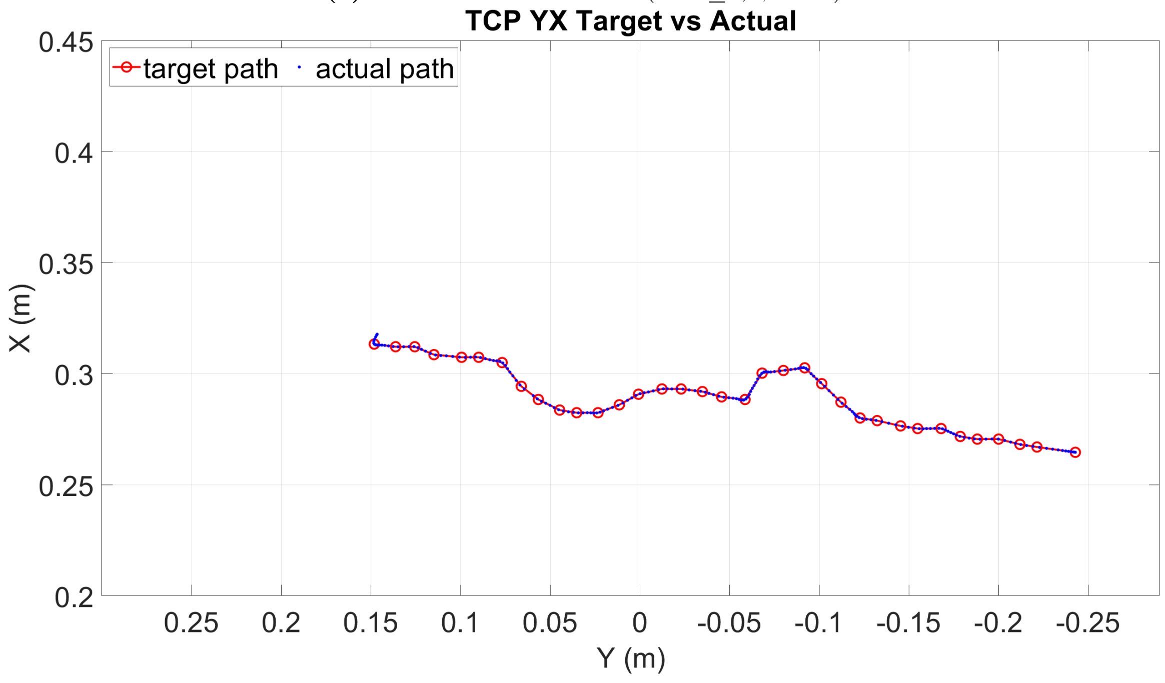
The execution times, seen in Figure 4.2, shows promising results. The average execution time is 3.365 seconds, and the average TCP speed is 116.3 $mm\ s^{-1}$. A study [63] explains that exposure times of 1-2 seconds represent highly relevant conveyor speeds and that the exposure time of 3-4 seconds is critical for solid results. The studies by Fenelon et al. [31] present in-line measurements with conveyor speeds ranging from 10 $mm\ s^{-1}$ to 100 $mm\ s^{-1}$.

Given the speed of 100 $mm\ s^{-1}$, the fillet will move about 300 mm in 3 seconds, which is inside the workspace of the UR3 arm, and it would be able to scan the fillet successfully. If the conveyor speed increases to 150 $mm\ s^{-1}$, the arm will risk failing, as it would move 450 mm in 3 seconds. Lastly, if the conveyor speed is 200-250 $mm\ s^{-1}$, it can be envisioned that the arm will fail since the salmon moves significantly quickly in, through, and out of the workspace. To solve this, an arm with a larger workspace and higher TCP speed, or a more efficient implementation of a path planner is required.

The fastest and slowest times are 2.56 s and 3.94 s for (fillet_0, $\psi = 0°$) and (fillet_2, $\psi = 0°$), respectively. The two results are shown in Figure 4.32, and it is no surprise that the fastest execution time came from the almost straight scan profile for fillet_0 (seen in Figure 4.32a). However, the slowest execution time for fillet_2 (seen in Figure 4.32b), might be due to the quite sharp turn at $y \approx -0.05, x \approx 0.3$, which is also the case for fillet_3. It is hard to say if that is the reason, since the execution also shows the same 'tail', which might affect the time measurements, as well as the few test samples. Overall, the execution times are relatively close, with the number of melanin spots showing the expected trend of increasing the execution time.



(a) Fastest Executed Path (fillet_0, $\psi = 0°$)



(b) Slowest Executed Path (fillet_2, $\psi = 0°$)

Figure 4.32: Fastest- and Slowest Executed Paths

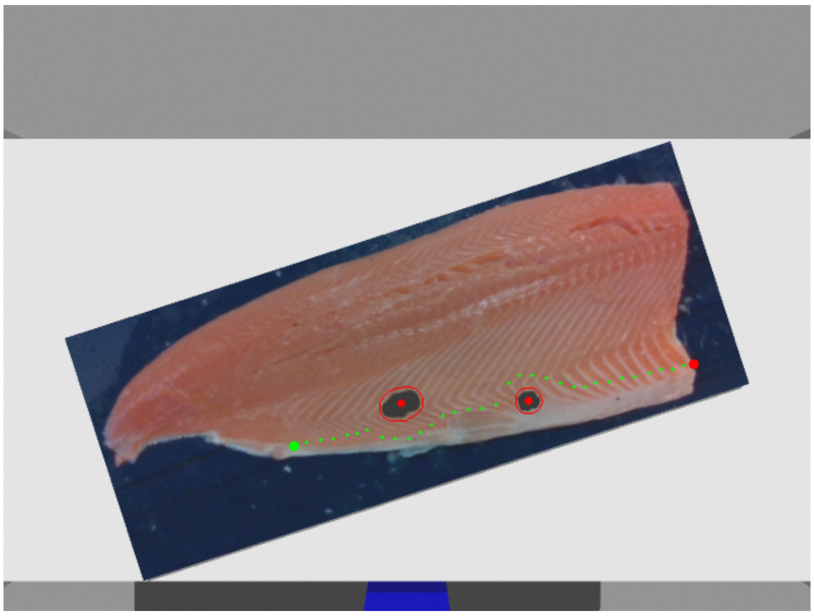


Figure 4.33: Fully Planned Path for fillet_2, $\psi = 18°$

Unsuccessful Execution

The arm performs consistently, and it was not many cases where it was not able to complete the whole scan. However, there was one case where the planner did not fully complete the planned path. Even though the planned path reaches the goal, as seen in Figure 4.33, the actual execution does not reach the end, as seen in Figure 4.34 and 4.35. The more detailed plot (Figure 4.35), also shows a dense cluster of trajectory waypoints toward the end, but the reason it did not complete the full plan is not really understood.

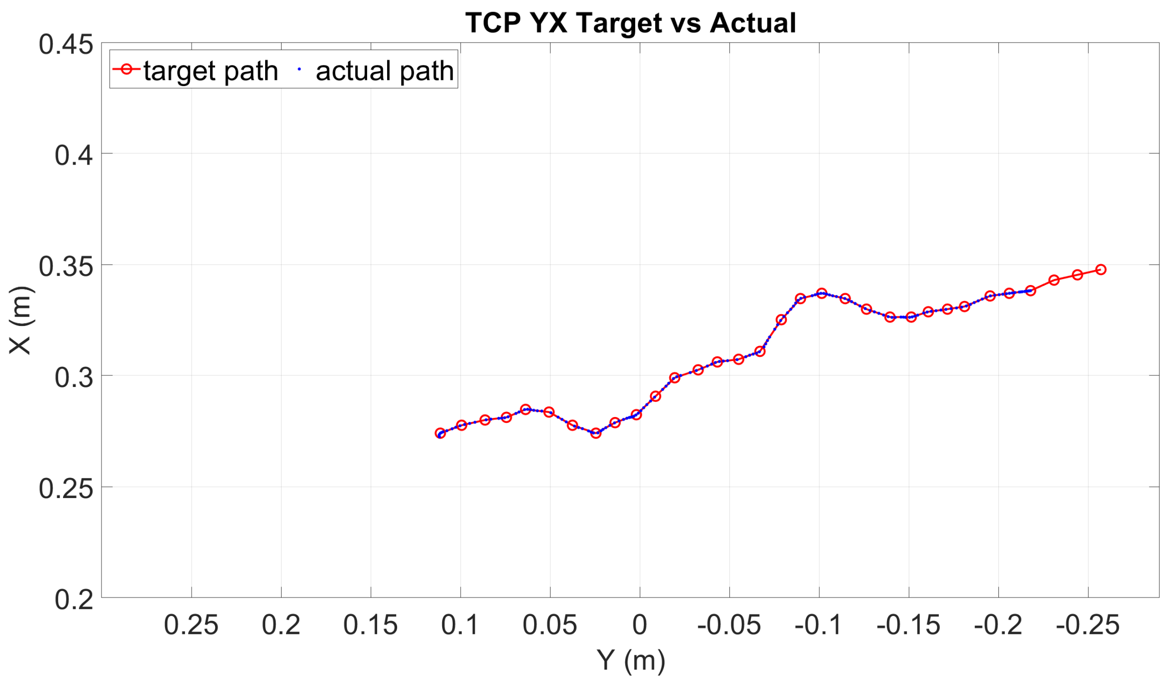


Figure 4.34: Unsuccessful Scan for fillet_2, $\psi = 18°$

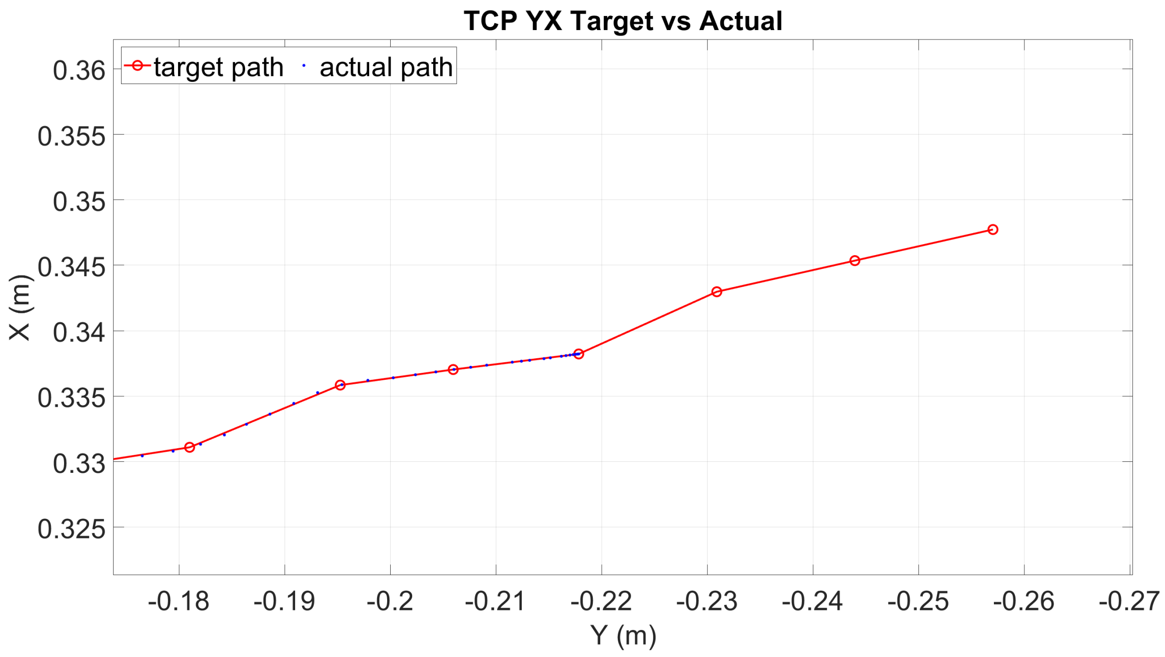


Figure 4.35: Unsuccessful Scan for fillet_2, $\psi = 18°$

The observation also brings in another topic for discussion, which is the distribution of waypoints (not trajectory points) for the arm. For a simple, close to straight scan profile, it is quite redundant to have many waypoints aligned along this profile, instead it is is more appropriate to have denser waypoints for more complex parts of the scan profile, like when the direction changes more rapidly. This is something that should be considered, for example, by checking the angle between subsequent waypoints in the planned path and removing excessive waypoints which do not contribute to a significant change in direction.

Chapter 5

Conclusion and Future Work

This chapter presents concluding remarks on the results obtained in this master's thesis and perspectives on the work developed here.

5.1 Conclusion

The aim of this thesis was to build a robotic Raman Spectroscopy system in simulation, more specifically, by addressing the research question stated in Chapter 1: "How can machine vision, AI-driven models, image processing, and path planning be used to generate a scan profile and instruct a robot arm to perform an effective scanning of salmon fillets, ensuring the avoidance of melanin spots?". This was addressed by introducing the robotic pipeline which was built throughout Chapter 3, Methodology, and the results are very promising. The system uses a deep learning-powered solution, as well as image processing techniques to retrieve the required information on the fillet, before the developed planner generates a path based on this information.

The main limiting factor in the presented work was the availability of the dataset of salmon with melanin spots, which should be used for training the AI model, as well as testing the performance of the new APF-like planner. This dataset was not accessible during the course of this work, so a previous dataset had to be used to generate a new dataset of salmon fillets with pseudo-melanin spots to mimic the unavailable data. This resulted in a sparse dataset, limiting the training and testing of the developed AI model and the path planner.

Using this dataset, the AI model is predominantly able to segment the binary mask of the belly, however, as discussed in Section 4.1.2, it might need some improvement due to somewhat poor performance in particular scenarios. Another approach can also be considered, like using object detection to create a bounding box instead of a segmentation, which can also make the image processing less complex. The evaluation of the AI-driven model's performance on the melanin spots is limited by the dataset availability. It does show solid performance on the pseudo-melanin spots. However, it still needs to be tested on real ones to get an insight into its ability to segment these image features. In addition, it is unlikely that the model training and performance on real spots will be a challenge, considering the promising AI performance on the belly, which is a less distinct area than the discolored melanin spots.

After tuning the parameters for the path planning algorithm, as well as evaluating different approaches and tuning parameters for the robot movement, the APF-like planner shows robust performance. It was successfully able to adapt to the different number of pseudo-melanin spots and fillet orientation, resulting in complex paths and robot trajectories. The robot end-effector is then able to accurately follow these trajectories in a simulated environment with an average execution time of 3.365 seconds, corresponding to an average TCP speed of 116.3 $mm\ s^{-1}$, which seems promising for the industry, as discussed in Section 4.3.1.

Based on the work developed in this thesis and the results, it is suggested to further modify the path planner to remove redundant waypoints in the resulting path. The most prominent example of this was seen in fillet_0, where the APF-like planner generates a relatively highly detailed path (Section 4.2.3), forcing the robot end-effector to go through these points, as seen in Section 4.3.1. In this example, only two points are necessary to efficiently follow the desired path, being the two extreme points. By solving this problem, for example, by removing points that do not contribute to a significant direction change, the robot arm will have fewer waypoints to visit, and will be able to achieve faster execution times.

5.2 Future Work

To further elevate the concept presented in this work, a more relevant dataset featuring salmon fillets with real melanin spots will give a better opportunity to more precisely evaluate the system. The next steps also include taking the system from the simulated environment, to using the real robot arm, natural salmon and Raman sensor in a relevant laboratory environment. This can lead to further tuning of the planner based on resulting Raman spectra. Finally, by using a moving conveyor belt and streamlining this process, the system would be one step closer to the industrial conditions. By successfully achieving the deployment of a fully autonomous robotic solution, food quality assessment in processing lines will become less error-prone, more cost-effective, and highly efficient. Although this work presented melanin spots as obstacles in the measurement of Omega-3 fatty acids, the melanin spots can also be a region of interest. For example, the chemical content or compounds in a melanin spot could lead to the cause of such spots. Similar visual planning and path planning can be used to plan to one or more melanin spots and measure spectra using a Raman spectroscopy or other sensors.

Bibliography

[1] Fiskeridirektoratet, "Akvakulturstatistikk matfiskproduksjon av laks, regnbueørret og ørret," `https://www.fiskeridir.no/Akvakultur/Tall-og-analyse/Akvakulturstatistikk-tidsserier/Laks-regnbueoerret-og-oerret/Matfiskproduksjon`, Accessed: 2024-01-16.

[2] T. Ytrestøyl, M. Bou, C. Dimitriou, G. M. Berge, T.-K. Østbye, and B. Ruyter, "Dietary Level of the Omega-3 Fatty Acids EPA and DHA Influence the Flesh Pigmentation in Atlantic Salmon," *Aquaculture Nutrition*, vol. 2023, pp. e5528942, Mar. 2023, Publisher: Hindawi.

[3] Turid Mørkøre, Thomas Larssson, Agnar Kvellestad, Erling Koppang, Magnus Åsli, Aleksei Krasnov, Jens-Erik Dessen, Helena Moreno, Elin Valen, Kjellrun Gannestad, Bjarne Gjerde, Torunn Taksdal, Grete Bæverfjord, Yuqiong Meng, Karsten Heia, Jens Wold, A.J. Borderías, Hooman Moghadam, Odd Romarheim, and Kjell-Arne Rørvik, *Mørke flekker i laksefilet. Kunnskapsstatus og tiltak for å begrense omfanget*, ResearchGate, Oct. 2015.

[4] NMBU-Norges miljø-og biovitenskapelige universitet and Mette Risbråthe, "Løste gåten med de mørke flekkene i laks," `https://www.forskning.no/partner-fiskesykdommer-oppdrett/loste-gaten-med-de-morke-flekkene-i-laks/473741`, Sept. 2015, Section: naturvitenskap, Accessed: 2024-05-03.

[5] Redaksjon, "Flere årsaker til melaninflekker i laks," `https://www.kyst.no/melanin-nmbu/flere-arsaker-til-melaninflekker-i-laks/1477880`, Jan. 2023, Accessed: 2024-05-03.

[6] Franziska Färber, "Melanin spots in Atlantic salmon fillets : an investigation of the general problem, the frequency and the economic implication based on

an online survey," M.S. thesis, Norwegian University of Life Sciences, Ås, 2017, Accepted: 2017-08-16T10:33:17Z Publication Title: 84.

[7] K. Eder, "Gas chromatographic analysis of fatty acid methyl esters," *Journal of Chromatography B: Biomedical Sciences and Applications*, vol. 671, no. 1, pp. 113–131, Sept. 1995.

[8] Nils Kristian Afseth, Katinka Dankel, Petter Vejle Andersen, Gareth Frank Difford, Siri Storteig Horn, Anna Sonesson, Borghild Hillestad, Jens Petter Wold, and Erik Tengstrand, "Raman and near Infrared Spectroscopy for Quantification of Fatty Acids in Muscle Tissue—A Salmon Case Study," *Foods*, vol. 11, no. 7, pp. 962, Jan. 2022, Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[9] Tiril Aurora Lintvedt, Petter Vejle Andersen, Nils Kristian Afseth, Karsten Heia, Stein-Kato Lindberg, and Jens Petter Wold, "Raman spectroscopy and NIR hyperspectral imaging for in-line estimation of fatty acid features in salmon fillets," *Talanta*, vol. 254, pp. 124113, Mar. 2023.

[10] John Reidar Mathiassen, Ekrem Misimi, and Amund Skavhaug, "A Simple Computer Vision Method for Automatic Detection of Melanin Spots in Atlantic Salmon Fillets," in *International Machine Vision and Image Processing Conference (IMVIP 2007)*, Sept. 2007, pp. 192–200.

[11] Michael A. A. Fenelon, Tiril A. Lintvedt, Jens P. Wold, and Antonio C. Leite, "A Robot and Sensor Integration Method to Measure Fatty Acid Composition in Salmon Fillets*," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, Aug. 2023, pp. 1–8, ISSN: 2161-8089.

[12] Bruno Siciliano and Oussama Khatib, Eds., *Springer Handbook of Robotics*, Springer Handbooks. Springer, 2016.

[13] Peter Corke, *Robotics, Vision and Control*, vol. 118 of *Springer Tracts in Advanced Robotics*, Springer International Publishing, Cham, 2017.

[14] Zhengyou Zhang, "A Flexible New Technique for Camera Calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 1330–1334, Dec. 2000.

[15] Antonio C Leite, Fernando Lizarralde, Pål Johan From, Ramon R Costa, and Liu Hsu, "Remote Calibration and Trajectory Replanning for Robot Manipulators

Operating in Unstructured Environments," *IFAC Proceedings Volumes*, vol. 45, no. 8, pp. 59–65, 2012.

[16] B. T. Polyak, "Newton's method and its use in optimization," *European Journal of Operational Research*, vol. 181, no. 3, pp. 1086–1096, Sept. 2007.

[17] Finn Aakre Haugen, "Modeling, Simulation and Control," `https://techteach.no/control/modsimcon/mod\_sim\_con\_2023\_08\_24.pdf`, Aug. 2023, Accessed: 2024-04-19.

[18] Michael A. Johnson, Mohammad H. Moradi, and J. Crowe, *PID control: new identification and design methods*, Springer, New York, 2005, OCLC: 65196439.

[19] Bernard Friedland, *Control System Design: An Introduction to State-Space Methods*, Courier Corporation, Mar. 2012, Google-Books-ID: 9WRKZlaCnF8C.

[20] Joe Qin and Thomas Badgwell, "An Overview Of Industrial Model Predictive Control Technology," *AIChE Symposium Series*, vol. 93, Jan. 1997.

[21] Ravi Teja, "Open Loop System," `https://www.electronicshub.org/open-loop-system/`, apr 2024, Accessed: 2024-04-21.

[22] Sebastian Raschka and Vahid Mirjalili, *Python Machine Learning*, Packt Publishing Ltd., Livery Place 35 Livery Street Birmingham B3 2PB, UK, second edition, Sept. 2017.

[23] David A van Dyk and Xiao-Li Meng, "The Art of Data Augmentation," *Journal of Computational and Graphical Statistics*, vol. 10, no. 1, pp. 1–50, Mar. 2001, Publisher: Taylor & Francis _eprint: https://doi.org/10.1198/10618600152418584.

[24] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo, *Robotics: Modelling, Planning and Control*, Advanced Textbooks in Control and Signal Processing. Springer London, London, 2009.

[25] Oussama Khatib, "The Potential Field Approach And Operational Space Formulation In Robot Control," in *Adaptive and Learning Systems*, Kumpati S. Narendra, Ed., pp. 367–377. Springer US, Boston, MA, 1986.

[26] Giuseppe Oriolo, "Autonomous and mobile robotics, lecture slides 3," `https://www.diag.uniroma1.it/oriolo/amr/slides/MotionPlanning3_Slides.pdf`, Accessed: 2024-04-03.

[27] Lixing Liu, Xu Wang, Xin Yang, Hongjie Liu, Jianping Li, and Pengfei Wang, “Path planning techniques for mobile robots: Review and prospect,” *Expert Systems with Applications*, vol. 227, pp. 120254, Oct. 2023.

[28] A Nasuha, A S Priambodo, and G N P Pratama, “Vortex Artificial Potential Field for Mobile Robot Path Planning,” *Journal of Physics: Conference Series*, vol. 2406, no. 1, pp. 012001, Dec. 2022.

[29] Xiaoping Yun and Ko-Cheng Tan, “A wall-following method for escaping local minima in potential field based motion planning,” in *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR’97*, July 1997, pp. 421–426.

[30] Rafal Szczepanski, Tomasz Tarczewski, and Krystian Erwinski, “Energy Efficient Local Path Planning Algorithm Based on Predictive Artificial Potential Field,” *IEEE Access*, vol. 10, pp. 39729–39742, 2022, Conference Name: IEEE Access.

[31] Michael Angelo Amith Fenelon, Tiril Aurora Lintvedt, Antonio Candea Leite, and Jens Petter Wold, “Robotic Raman Spectroscopy for Rapid Fatty Acid Measurement in Salmon Fillets,” *IFAC-PapersOnLine*, vol. 56, no. 2, pp. 9757–9764, Jan. 2023.

[32] Stanford Artificial Intelligence Laboratory et al., “Robotic operating system,” `https://www.ros.org/`, Accessed: 2024-04-23.

[33] ROS Developers, “ROS/Tutorials/WritingPublisherSubscriber - ROS Wiki,” `http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29`, Accessed: 2024-05-12.

[34] N. Koenig and A. Howard, “Design and use paradigms for Gazebo, an open-source multi-robot simulator,” in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 2004, vol. 3, pp. 2149–2154 vol.3.

[35] ROS Developers, “urdf/Tutorials - ROS Wiki,” `https://wiki.ros.org/urdf/Tutorials`, Accessed: 2024-05-12.

[36] rokokoo, “gazebo-conveyor,” `https://github.com/rokokoo/gazebo-conveyor`, Accessed: 2024-05-12.

[37] Blender Foundation, “blender.org - Home of the Blender project - Free and Open 3D Creation Software,” `https://www.blender.org/`, Accessed: 2024-05-12.

[38] Intel RealSense, "Stereo Depth Solutions from Intel RealSense," `https://www.intelrealsense.com/stereo-depth/`, 2024, Accessed: 2024-04-10.

[39] Chia-Kai Liang, Li-Wen Chang, and Homer H. Chen, "Analysis and Compensation of Rolling Shutter Effect," *IEEE Transactions on Image Processing*, vol. 17, no. 8, pp. 1323–1330, Aug. 2008, Conference Name: IEEE Transactions on Image Processing.

[40] Intel RealSense, "Depth Camera D435 – Intel® RealSense™ Depth and Tracking Cameras," `https://www.intelrealsense.com/depth-camera-d435/`, 2023, Accessed: 2024-03-04.

[41] PAL Robotics S.L., "realsense_gazebo_plugin," `https://github.com/pal-robotics/realsense_gazebo_plugin`, 2024, Accessed: 2024-04-25.

[42] ROS Developers, "camera_calibration - ROS Wiki," `http://wiki.ros.org/camera_calibration`, Accessed: 2024-05-03.

[43] ROS Developers, "Align Depth ROS Topic · Issue #31 · pal-robotics/realsense_gazebo_plugin," `https://github.com/pal-robotics/realsense_gazebo_plugin/issues/31`, 2024, Accessed: 2024-04-25.

[44] Jiří Ulrich, Peter Lightbody, Aaron Weinstein, Filip Majer, and Tomáš Krajník, "WhyCode: Efficient and Versatile Fiducial Localisation System," Oct. 2018.

[45] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[46] ROS Developers, "tf - ROS Wiki," `http://wiki.ros.org/tf`, Accessed: 2024-04-23.

[47] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick, "Detectron2," `https://github.com/facebookresearch/detectron2`, 2019, Ac-

cessed: 2024-04-03.

[48] Kazumasa Wakamatsu, Johannes M. Dijkstra, Turid Mørkøre, and Shosuke Ito, "Eumelanin Detection in Melanized Focal Changes but Not in Red Focal Changes on Atlantic Salmon (Salmo salar) Fillets," *International Journal of Molecular Sciences*, vol. 24, no. 23, pp. 16797, Jan. 2023, Number: 23 Publisher: Multidisciplinary Digital Publishing Institute.

[49] Raúl Jiménez-Guerrero, Grete Baeverfjord, Øystein Evensen, Kristin Hamre, Thomas Larsson, Jens-Erik Dessen, Kjellrun-Hoås Gannestad, and Turid Mørkøre, "Rib abnormalities and their association with focal dark spots in Atlantic salmon fillets," *Aquaculture*, vol. 561, pp. 738697, Dec. 2022.

[50] PintaProject, "Pinta: Painting made simple," `https://www.pinta-project.com/howto/installing-pinta`, Accessed: 2024-04-29.

[51] Lerøy Seafood Group, "Fillets & portions," `https://www.leroyseafood.com/en/tasty-seafood/product-range/fillets--portions/`, Accessed: 2024-05-03.

[52] Abhishek Dutta and Andrew Zisserman, "The VIA Annotation Software for Images, Audio and Video," in *Proceedings of the 27th ACM International Conference on Multimedia*, New York, NY, USA, 2019, MM '19, ACM, event-place: Nice, France.

[53] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick, "Microsoft COCO: Common Objects in Context," in *European Conference on Computer Vision*, 2014, pp. 740–755.

[54] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[55] OpenCV, "Structural Analysis and Shape Descriptors," `https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#gaf849da1fdafa67ee84b1e9a23b93f91f`, Accessed: 2024-05-05.

[56] Universal Robots, "Universal Robots - User manual - UR3 CB-Series - SW3.3 - English international," `https://www.universal-robots.com/download/manuals-cb-series/user/ur3/33/user-manual-ur3-cb-series-sw33-english-international/`, 2021, Accessed: 2024-03-04.

[57] MarqMetrix, "MarqMetrix All-In-One spec sheet 8023 v3.2," https://marqmetrix.com/products/all-in-one/, 2021, Accessed: 2024-03-04.

[58] ROS Developers, "ur_gazebo - ROS Wiki," http://wiki.ros.org/ur_gazebo, Accessed: 2024-05-03.

[59] Moveit, "MoveIt Motion Planning Framework," https://moveit.ros.org/, Accessed: 2024-04-23.

[60] Moveit, "Robots | MoveIt," https://moveit.ros.org/robots/, Accessed: 2024-04-23.

[61] ROS Documentation, "move_group: move_group.cpp File Reference," http://docs.ros.org/en/noetic/api/moveit_ros_move_group/html/move__group_8cpp.html, Accessed: 2024-04-23.

[62] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sept. 2020, Publisher: Springer Science and Business Media LLC.

[63] Tiril Aurora Lintvedt, Petter V. Andersen, Nils Kristian Afseth, Brian Marquardt, Lars Gidskehaug, and Jens Petter Wold, "Feasibility of In-Line Raman Spectroscopy for Quality Assessment in Food Industry: How Fast Can We Go?," *Applied Spectroscopy*, vol. 76, no. 5, pp. 559–568, May 2022.