Design of Autonomous Robotic System For Removal of Porcupine Crab Spines

by

© *Haodong Wu*

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Engineering

Department of *Mechanical Engineering*

Memorial University of Newfoundland

May 2023

St. John's Newfoundland

Abstract

Among various types of crabs, the porcupine crab is recognized as a highly potential crab meat resource near the off-shore northwest Atlantic ocean. However, their long, sharp spines make it difficult to be manually handled. Despite the fact that automation technology is widely employed in the commercial seafood processing industry, manual processing methods still dominate in today's crab processing, which causes low production rates and high manufacturing costs.

This thesis proposes a novel robot-based porcupine crab spine removal method. Based on the 2D image and 3D point cloud data captured by the Microsoft Azure Kinect 3D RGB-D camera, the crab's 3D point cloud model can be reconstructed by using the proposed point cloud processing method. After that, the novel point cloud slicing method and the 2D image and 3D point cloud combination methods are proposed to generate the robot spine removal trajectory.

The 3D model of the crab with the actual dimension, robot working cell, and end-effector are well established in Solidworks [1] and imported into the Robot Operating System (ROS) [2] simulation environment for methodology validation and design optimization. The simulation results show that both the point cloud slicing method and the 2D and 3D combination methods can generate a smooth and feasible trajectory. Moreover, compared with the point cloud slicing method, the 2D and 3D combination method is more precise and efficient, which has been validated in the real experiment environment.

The automated experiment platform, featuring a 3D-printed end-effector and crab model, has been successfully set up. Results from the experiments indicate that the crab model can be accurately reconstructed, and the central line equations of each spine were calculated to generate a spine removal trajectory. Upon execution with a real robot arm, all spines were removed successfully. This thesis demonstrates the proposed method's capability to achieve expected results and its potential for application in various manufacturing processes such as painting, polishing, and deburring for parts of different shapes and materials.

Key Words: robotic manufacturing, trajectory planning, point cloud segmentation, ROS

Acknowledgements

I would like to express my gratitude to my supervisor, Dr. Zou, for her guidance and support in my master's program and for providing me with a good working and experiment environment during my research program. Without her patience, comments and encouragement, I would not finish this thesis smoothly and fruitfully.

I am also grateful for the help from Mr. Stephen King and Ms. Heather Burke from the Marine Institute (MI) and Mr. Brian Burke from the Nunavut Fisheries Association (NFA) for providing me with the opportunity and help to get hands-on materials of the Porcupine crab.

This work is supported by Mitacs and Nunavut Fisheries Association through the Mitacs Accelerate Award.

In addition, I would also like to thank my family for their support and encouragement.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the ongoing development trend of Industry 4.0 [3], nearly every major traditional manufacturing industry is proceeding automation improvement and introducing robot-based systems into their production process to improve productivity, quality and profitability. Compared with human workers, robots can adapt to different operating environments and fulfill various production requirements. Therefore, robots might become the main labour force in the future and replace human workers to proceed with tedious, repetitive and dangerous work.

Porcupine crab is chosen as this thesis's research object since it has high potential commercial value and is challenging to process. This thesis is concerned with establishing a robot-based system to identify, locate and remove the spines on the crab's body. It is envisaged that this system could be used in future porcupine crab production and could also potentially be extended to many other seafood production processes. The research on this topic can provide a case for the automation upgrade of the seafood processing industry. Additionally, the project is a great platform for enhancing my ability to design complete computer vision and robot automation systems, and for gaining valuable experience that can serve as a foundation for future work and research.

Individual crabs differ in body sizes and spine growing locations, which requires the robot-based spine removal system to be highly adaptable and flexible. A 3D RGB-D camera is used in this thesis to capture 2D image and 3D point cloud data that are processed using a 3D reconstruction algorithm to establish a 3D mesh model for each crab. Due to the limited resolution of low-cost cameras, it is difficult to accurately establish spine features. This thesis presents a novel method that combines 2D images and 3D point clouds to effectively identify and locate spines.

After acquiring all the coordinates of spines, a novel point cloud slicing method and a cubic curve-based trajectory planning method are introduced, the trajectory of the robot end-effector along with its associated poses thus being generated. The robot trajectory smoothly remove all the spines without interfering with other parts of the crab body, exceeding the operating range and payload limitation of the robot arm.

Before building a real robot experimental system, it is an ideal way to build the same platform inside a simulation environment with the same parameters as in the real world. Use of simulation allows one to verify the design of robots, sensors and other accessories before focusing on hardware development and incurring the associated cost. Also, it allows one to test the whole algorithm rapidly. This thesis selects the Robot Operating System (ROS) as the simulation platform, based on which the entire robotic system was built. The simulation results have verified the feasibility of the whole capturing and spine removal processes.

During the simulation process, the design is continuously iterated and optimized. Afterwards, the robot-based spine removal system was built in terms of the optimization results. The experimental test results showcase the feasibility of the proposed methodology.

1.1 Thesis Overview

This section shows the thesis structure in order to provide the reader with an insight into the main content of each chapter.

Chapter 1 gives an overview of the thesis, background and road map. It introduces the porcupine crab species and reviews the current state-of-art seafood crab processing methods, robot-based deburring applications, robot simulation environment and 3D reconstruction methods.

Chapter 2 describes the detailed design steps of the robot-based spine removal system, including the design philosophy, mechanical design of the robot working station and end-effector, electric and control design of the system, as well as the simulation and software environment setup.

Chapter 3 describes the proposed methodology, including the point cloud processing method, robot trajectory, and tool pose planning method. This chapter describes how to process the input 2D image and 3D point cloud data to acquire information about the spines and how to generate the robot spine removal trajectory.

Chapter 4 and **Chapter 5** present the simulation and real-world experimental results. Both the simulation and experimental results validate the design objectives.

Chapter 6 draws the conclusions of the thesis and discusses the possible future directions to pursue and the follow-up research work.

1.2 Literature Review

1.2.1 Porcupine Crab Overview and Analysis

According to the latest research progress in the deep-sea fisheries field, the porcupine crab(*Neolithodes grimaldii*) is a sub-species of the king crab family, that is defined as a high incidental by-catch in offshore Greenland and the Northwest Atlantic areas [4], the map of the species distribution being shown in Fig. 1.1. It is noteworthy that marketable crab meat products can be produced from the porcupine crab, which is attractive in future fisheries [5]. As shown in Fig. 1.2, as a distinctive appearance among the king crab family, the red long sharp spines of the porcupine carb cause tremendous difficulty in processing and thus being harmful to human workers. Therefore, removing the spines prior to processing becomes an urgent need.

For male and female crabs, no obvious differences exist in the size, weight, spine length, spine distribution and other characteristics, thus the same method for spine removal are applied for both [4]. The mean weight of the porcupine crab is about 2.28 kg, and the mean size is around 122.7 mm (CL) $\times$ 105.2 mm (CW) [1] [4], making it suitable to be processed by a robot with a light payload.

According to the on-site measurement results, the spines are evenly distributed at the surface of the crab's shell. As shown in Fig. 1.3, the length of the spines is between 15 mm to 45 mm, with a gradual decrease in length from the center to the

[1]The unit of carapace length(CL) and carapace width(CW) are used to describe the dimension of crab shell.

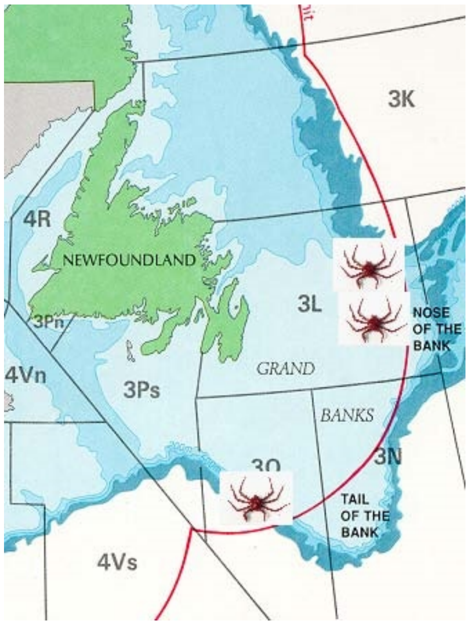


Figure 1.1: Distribution map of the porcupine Crab (*Neolithodes grimaldii*) [4]

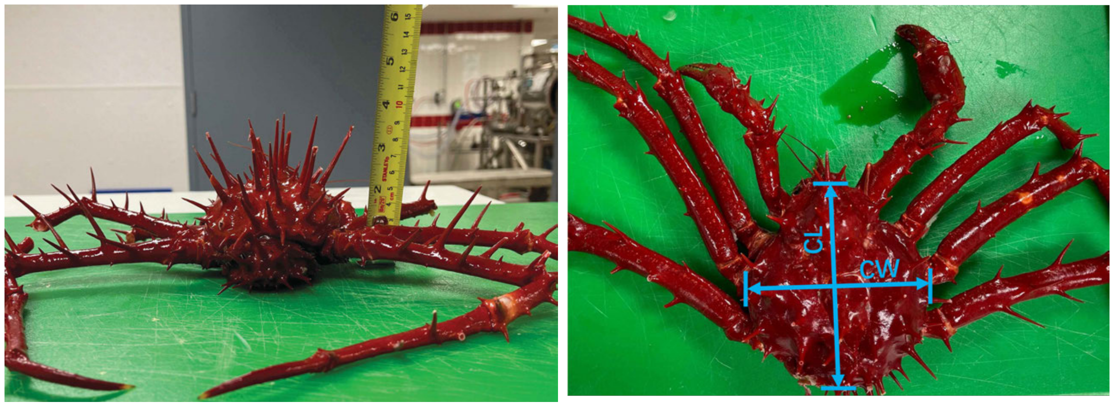


Figure 1.2: Images of the porcupine Crab (*Neolithodes grimaldii*)

edge of the shell. The diameter of the spine root is about 5 mm, with a smooth edge between the spine and the crab shell. Furthermore, the growing direction of each spine is perpendicular to the crab shell.

Chen and Lin demonstrated a puncture, tensile and compression test method for testing the hardness of king crab shells [6]. The crab shell hardness is approximately 60 D[2], which will be used for the cutting tool selection.

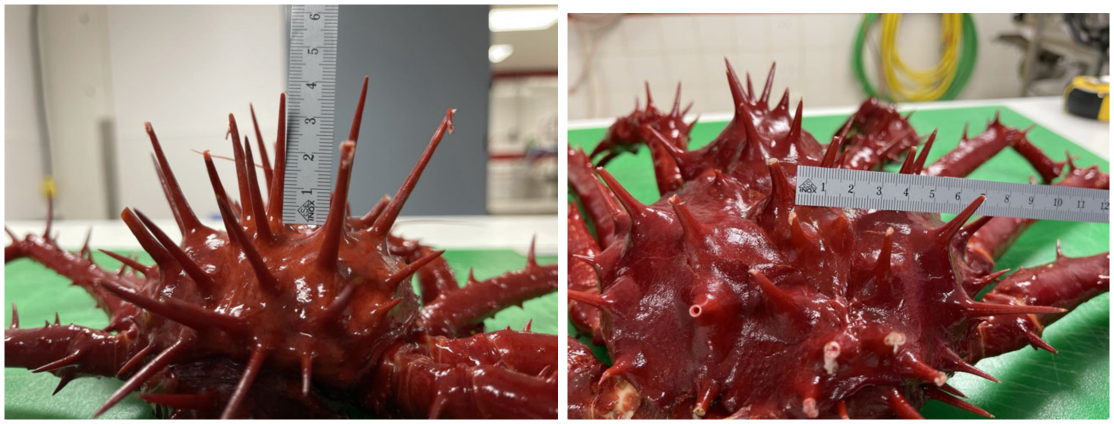


Figure 1.3: porcupine crab spine on-site measurement

1.2.2 Overview of Sea Food Automatic Processing Systems

In recent years, more and more automatic equipment has been used for massive production in seafood processing factories. As shown in Fig. 1.4, with the help of the latest technologies of computer vision, robotics and automation, the automatic salmon fish processing equipment [7, 8] has the ability to automatically identify and scale each salmon fish, calculate the best parameters to clean, slice the raw fish into saleable Sashimi.

As shown in Fig. 1.5, an interesting case of snow crab automatic butchering equipment is built by RYCO. Inc and BAADER. Inc [9, 10]. Raw snow crab can be automatically fastened, butchered and cleaned into saleable clusters. The usages of

[2]Shore Hardness is a standardized hardness test method. The hardness value is determined by the penetration of the durometer indenter foot into the sample.

automation, hydraulic and pneumatic technologies empower the equipment to process more than 42 crabs per minute, significantly reducing labour requirements and increasing productivity.

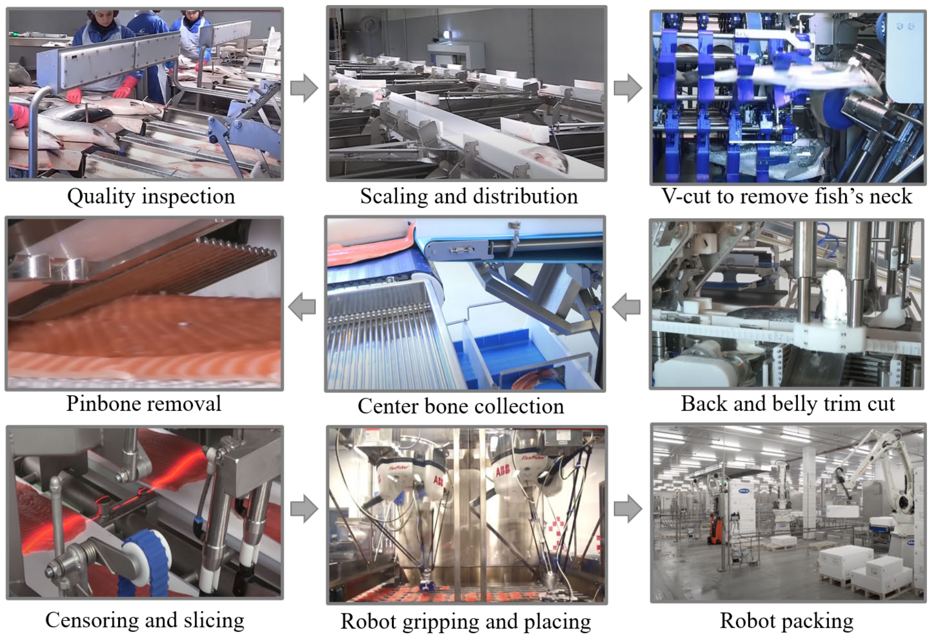


Figure 1.4: Salmon fish automatic processing equipment [8]

Although the systems mentioned above can process fish and crabs on a large scale, they still have several shortcomings. For example, they can only process one species and requires an enormous installation space. As shown in Fig. 1.6, a sensor-guided robot automated crustaceans processing system was developed by King and Hearn [11, 12]. This system can be used to process the snow crab, lobster and many other crustaceans with less installation space. It has the ability to intelligently identify the input categories of crustaceans and acquire the physical characteristics of each crustacean with the sensors installed. Based on the sensor data, one five-axis delta

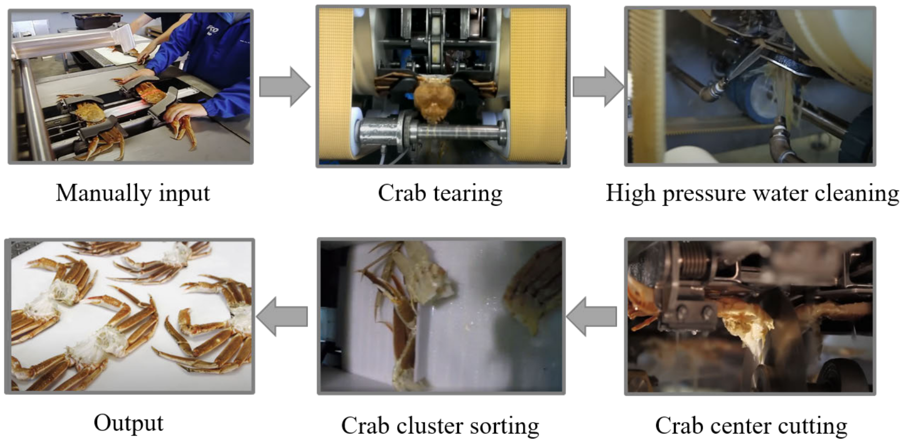


Figure 1.5: Snow crab automatic processing equipment [9]

robot picking the crustacean and placing it into a holding system with the optimal fixed position. Two six-axis robots cut the crustacean to produce optimal crustacean portions.

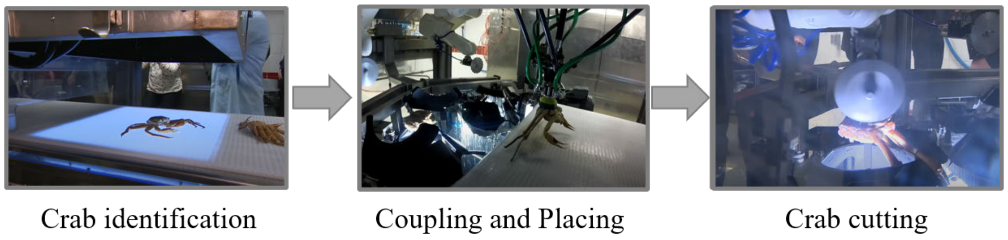


Figure 1.6: Sensor-guided robot automated crustaceans processing system [11]

In addition to the equipment mentioned earlier, other types of automated equipment are also commonly used in the seafood industry. However, a drawback of this equipment is that it can only process species with minimal differences in size and characteristics among individuals. Despite this, manual labor is still heavily relied upon for processing king crab. As depicted in Fig. 1.7, manual processing of king crab [13, 14] involves holding the crab's shoulders and striking it against a stationary anvil-like device on a butchering table. However, the spines on porcupine crab make it difficult to process and can cause injury to workers. Therefore, spine removal is a crucial step before further processing.

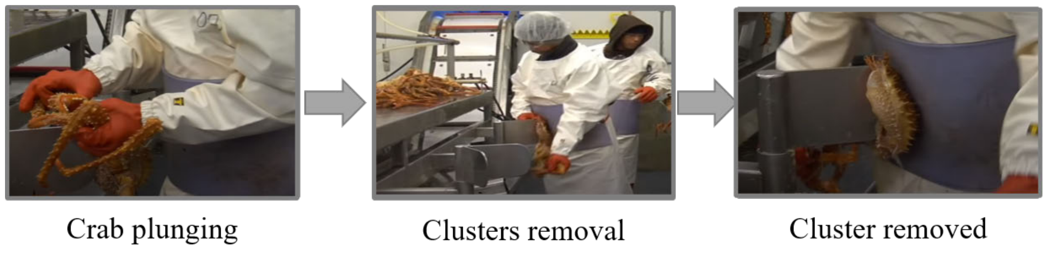


Figure 1.7: King crab manually processing method [13]

1.2.3 Overview of Robot-based Deburring Systems and Trajectory Planning Methods

1.2.3.1 Robot-based Deburring Systems

The robot-based spine removal system uses a similar process to the robot deburring system, making the latter a valuable reference point. The deburring is a widely used mechanical processing applications. It can be used for treating not only the metallic and non-metallic materials, but also for the other difficult-to-process materials such as polymers and composite materials. There are three main deburring methods in today's machining industry, including manual deburring, CNC deburring and robot deburring [15]. As shown in Fig. 1.8, manual deburring is most commonly used but may suffer from the disadvantages of low-efficiency and laborious. Also, the working environment is full of dust and noise, seriously affecting the workers' health [16]. In addition, it is hard to control manual operations' product consistency and quality. CNC and robot deburring methods are two other commonly used methods, which have been applied in many industrial sectors. CNC deburring machines have the advantages of high stiffness, production efficiency, accuracy and reliability but are also expensive and can only deburr workpieces on small scales (up to 600mm [17]).

(a) Manual deburring method

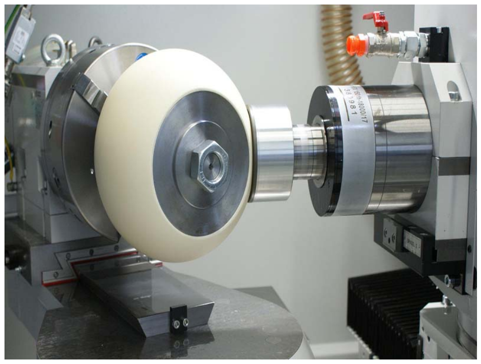
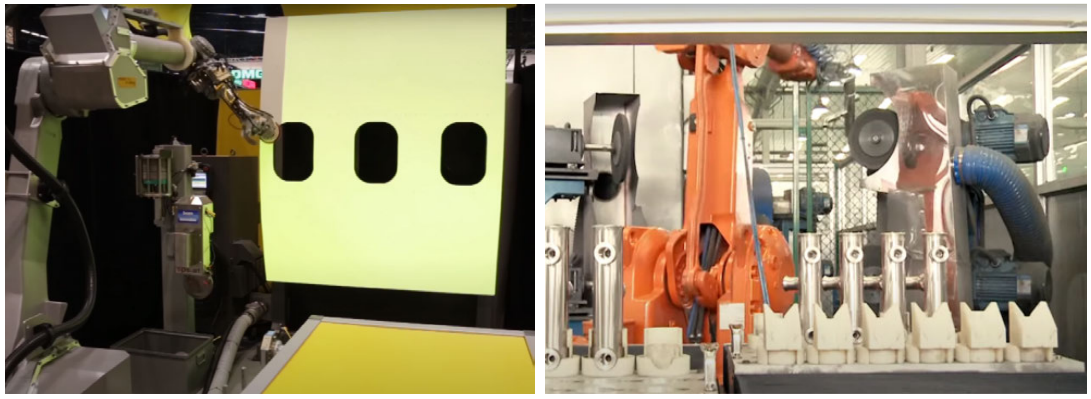

(b) CNC deburring method

Figure 1.8: Manual and CNC deburring method [16]

Compared with other deburring methods, the robot deburring method exhibits many advantages. One of the advantages is the low cost, the price of the robot arms is 60% - 80% lower than that of the CNC machines [18]. Besides, the six-axis robot arms have a larger working range (up to 3500 mm) compared with the CNC machines (up to 1500 mm), which allows the workpieces be in a larger scale. Also the serial mechanism of the robot arm gives it the ability to change the position and pose flexibly to ensure a better relative position between the deburring tool and part, which can improve the deburring accuracy [19, 20]. With the development of censoring and measuring devices [21], robots can be easily integrated with many intelligent sensors and cameras, such as 3D measurement cameras [22], force-torque sensors [23], to intelligently adjust the deburring parameters in real time.

As shown in Fig. 1.9 [24, 25], current state-of-art robot-based deburring systems can be divided into two types. The first one is to install the deburring tool onto the robot flange as the end-effector, the robot then carries the deburring tool to deburr the external fixed workpiece. This method is usually used when the robot payload is low, or when the workpiece is difficult to clamp, such as the car bodies [26], aircraft fuselage [27], and the porcupine crab in this thesis also belong to this situation. Another one is that the robot grips the workpiece with a gripper and performs the deburring operations on an external belt sander. This method is usually used when the part is on a small scale, such as the water faucets [28] and motorcycle helmets. etc.



(a) Robot fuselage deburring system (b) Robot faucet deburring system

Figure 1.9: Robot fuselage and faucet deburring systems [24, 25]

1.2.3.2 Robot Deburring Trajectory Planning Methods

Existing robot-based deburring systems are commonly used for deburring objects with regular or consistent shapes and features. Moreover, the robot trajectories are mainly generated by manual teaching [29], in which a robot operator manually moves the robot and records the robot path with a teach pendant or other similar devices. In case the deburring trajectory comprises a large number of curves, it is a common practice to approximate it in terms of a series of straight line segments, giving rise to the increase of number of way-points to be programmed. This method may lead to time-consuming and unsatisfactory effects, and also the robot operator must be experienced to manually teach way-points with a proper point density and step length [30].

Another widely used trajectory planning method for robot deburring relies on the accurate modelling of the object shape, which is pre-established inside a 2D or 3D structure design software. Some commercial software packages such as MasterCAM [31], Delmia [32], ABB RobotStudio [33] and others provide the function of robot deburring trajectory planning to follow the shape of the object's 3D model. The robot deburring trajectory can be planned, simulated and visualized in the aforementioned software. However, this feature highly relies on the identical object shape of the pre-defined model.

In order to meet the growing demands form the industry to scan and deburr parts with irregular shapes, the point cloud based robot deburring trajectory planning method has become a hot research direction in recent years [34, 35]. Using the point cloud data acquired with the 3D camera or the laser scanner, researchers proposed a method to generate B-spline curve tool path for three-axis milling machine, which was verified in Matlab [36]. The point cloud segmentation method for robot deburring trajectory planning also has a significant potential in manufacturing process. For example, with the combination of the point cloud slicing and smoothing method, an optimization approach based on point cloud was developed to ensure that the grinding tool and the work piece are in the maximum contact rate [37]. In order to increase the efficiency for grinding trajectory planning, the point cloud data can be projected onto a 2D plane to plan the tool path, then the tool path is projected back to the 3D shape [38]. In spite of substantial studies on the robot-based deburring trajectory planning, current state-of-the-art research mainly focuses on objects either with regular shapes or rigid parts, which eases the modeling process prior to deburring. Unfortunately, there is no common way to generate a robot deburring trajectory to process soft objects and objects with varied shapes, the seafood processing being a typical challenge.

1.2.4 Overview of Robot and Robot Operating System (ROS)

1.2.4.1 Robot Background Analysis

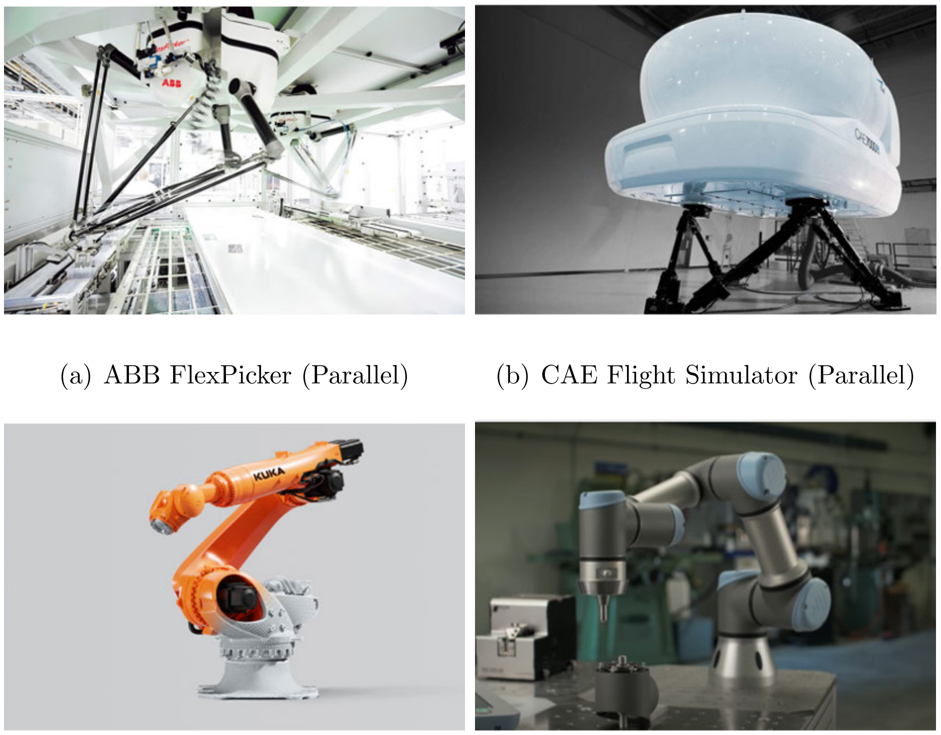
Robot is a highly-integrated system, an interdisciplinary branch of materials science, mechanical engineering, electrical engineering and control technology. International standardization organization (ISO) defines the robot as an actuated mechanism programmable in more than two axes with a degree of autonomy, moving in a designed environment, and performing the desired tasks [39]. Current state-of-art robot systems are designed to help and replace humans doing tedious, high-intensity and dangerous tasks with a high accuracy [40].

Robots can be categorized in terms of different criteria. Here we take the kinematic structure as an example, according to which, robots are classified into the parallel robot and serial robot [41], with some typical examples shown in Fig. 1.10. The parallel robot consists of several computer-controlled serial links connected at the coincident point of a single platform. Compared with the serial robot, each link of the parallel robot is simple, short and rigid, only supporting the fraction weight of the total load weight, which gives the parallel robot a heavy payload. Parallel robots are mostly used for pick-and-place processes, as well as the flight and ship simulators.

The serial robot consists of a series of links connected by motor-actuated joints that extend from the robot base to the robot flange (end-effector). The six-axis serial robot is the most popular model because having six degrees of freedom (DOFs), allows it to reach nearly every point in its working range with multiple poses. The most popular application for serial robots in today's industry includes welding, deburring, painting, pick-and-place.

As shown in Fig. 1.11, a six-axis collaborative robot[3] is used in this thesis. According to the EN/ISO 10218 (robot and robotic devices – safety requirements for industrial robots), traditional industrial robots are required to be installed far away from human operators in a separate working space because fast-running heavy-load robots can cause severe injury or even death for workers [46]. On the other hand, the cobot is designed and developed to have the ability to work with human workers shoulder-by-shoulder [47], whose safety relies on the usage of lightweight materials, smooth surfaces with round edges, speed and force limitations as well as software and sensors that make the cobot safe to human [46, 48], collaborative robots are ideal for

[3]Collaborative robot is a robot that is used for direct human-robot interaction in a shared space.



(a) ABB FlexPicker (Parallel) (b) CAE Flight Simulator (Parallel)

(c) KUKA Robot (Serial) (d) UR Robot (Serial)

Figure 1.10: Examples of parallel and serial robots [42, 43, 44, 45]

use in the seafood processing industry and in laboratory environments.

1.2.4.2 Robot Operating System (ROS)

The robot control and simulation technologies are constantly updated to meet the changing requirement in today's industrial and academic environments. Due to the rapid iteration of the electronic and control technologies, it becomes increasingly urgent to have a unified robot development platform on which robotics researchers can quickly develop and verify their research work and collaborate with other researchers worldwide. As shown in Fig. 1.12, the problem has been solved by the emergence of the open-source Robot Operating System (ROS) [49], as a highly extensible and pop-

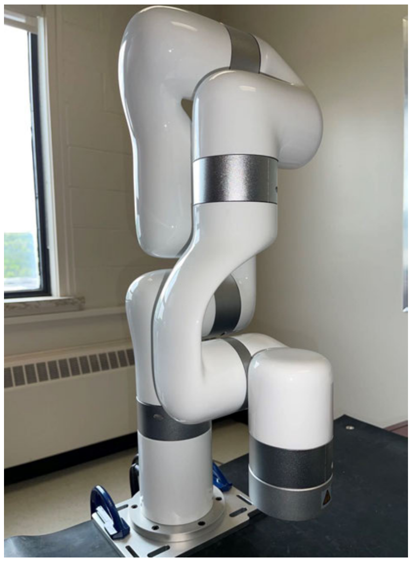


Figure 1.11: Xarm6 collaborative robot

ular distributed robotic development platform, ROS supports multiple development languages, including C++, Python, Java, Lisp. Many state-of-art function packages have been developed on this platform which significantly help the developer to improve their development efficiency and decrease the development difficulties. As a result, ROS is becoming more and more popular among international robotics researchers.

As an open-source robot control and simulation platform, ROS allows researchers to make personalized changes to the source codes as their needs and upload their code to the ROS community and various platforms, such as ROS Wiki [2], Github [51] and DockerHub [52]. Currently, ROS includes a lot of useful libraries for controlling robots as an operating system, researchers can search for the software development results from different researchers from all over the world, which provides convenience for robotics research.

Many commonly used hardware and software function packages have been en-

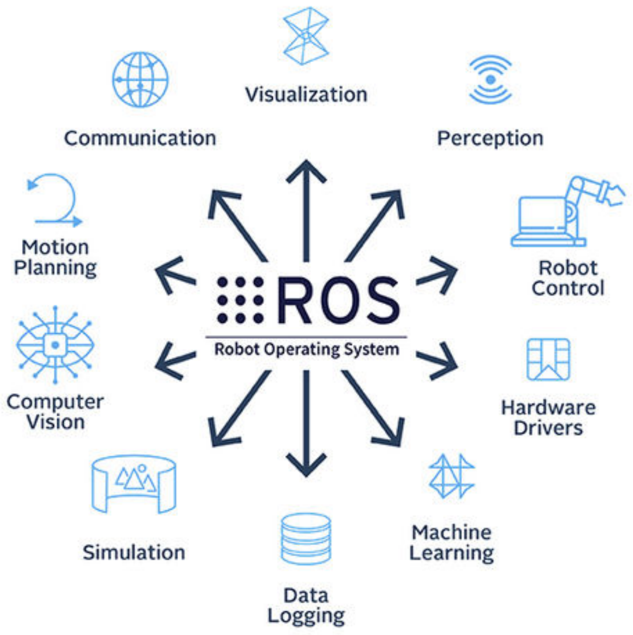
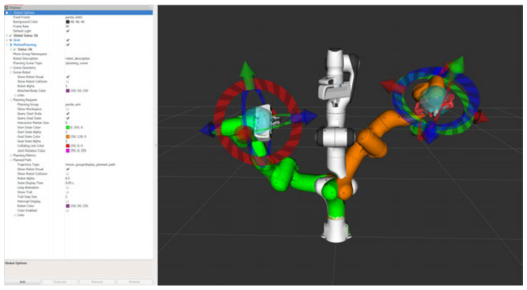


Figure 1.12: The architecture of the robot operating system (ROS) [50]

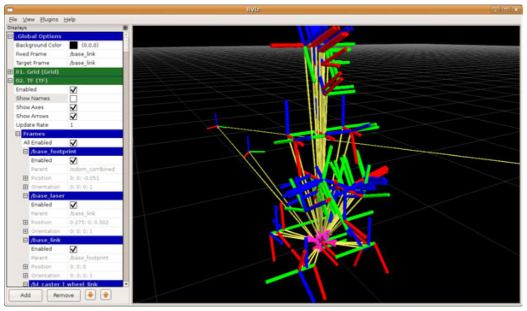
capsulated on the ROS platform, such as sensor drivers, robot trajectory planning packages and robot motion control packages. These packages contain ROS libraries, data sets, configuration files, source code, definition files, CMake build files, starting up files, parameter value sets, document files and other ROS-related files. ROS has also integrated many useful third-parties tools and libraries to help users and developers quickly build, test and commission their robot applications.

ROS RViz (ROS visualization) [53] is a powerful 3D visualization tool for robots, sensors and algorithms. It enables users to visualize the robot's state, trajectory, sensor data, and gives users an intuitive way to see the system operating status in real-time. Figure 1.13 [53] shows the typical functions of ROS RViz.

As shwon in Fig. 1.14 [54], ROS Gazebo [54] is a 3D dynamic simulator capable of accurately and efficiently simulating indoor and outdoor environments with gravity, friction and lighting parameters. Like the game developing engines, ROS Gazebo provides high-fidelity visual and physical simulations with a full set of sensor models and user-friendly interactive functions.

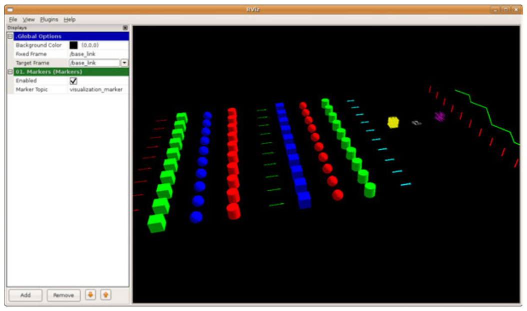


(a) Robot model display



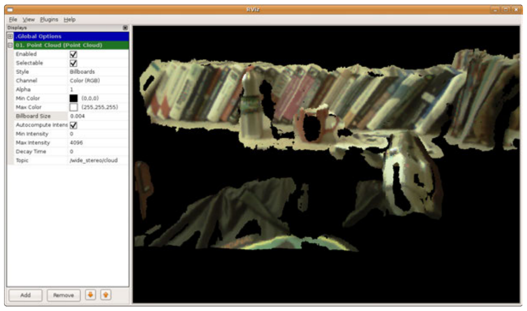
```
.Global Options
  Background Color     (0,0,0)
  Fixed Frame          /base_link
  Target Frame         /base_link
01. Grid (Grid)
02. TF (TF)
  Enabled              ✓
  Show Names
  Show Axes            ✓
  Show Arrows          ✓
  Update Rate          1
  Frames
    All Enabled        ✓
    /base_footprint
      Enabled          ✓
      Parent           /odom_combined
      Position         0; 0; -0.051
      Orientation      0; 0; 0; 1
    /base_laser
      Enabled          ✓
      Parent           /base_link
      Position         0.275; 0; 0.302
      Orientation      0; 0; 0; 1
    /base_link
      Enabled          ✓
      Parent           /base_footprint
      Position         0; 0; 0
      Orientation      0; 0; 0; 1
    /bl_caster_l_wheel_link
```

(b) Robot links & joints display



```
.Global Options
  Background Color     (0,0,0)
  Fixed Frame          /base_link
  Target Frame         /base_link
01. Markers (Markers)
  Enabled              ✓
  Marker Topic         visualization_marker
```

(c) Visualization markers



```
.Global Options
01. Point Cloud (Point Cloud)
  Enabled              ✓
  Selectable           ✓
  Style                Billboards
  Channel              Color (RGB)
  Alpha                1
  Min Color            (0,0,0)
  Max Color            (255,255,255)
  Billboard Size       0.004
  Autocompute Intens   ✓
  Min Intensity        0
  Max Intensity        4096
  Decay Time           0
  Topic                /wide_stereo/cloud
```

(d) 2D Image & 3D Point cloud display

Figure 1.13: ROS RViz typical functions [53]

Many robots were developed based on the ROS system, the first and most famous is the PR2 robot, which was developed by Willow Garage [55]. The PR2 robot consists of an omnidirectional base, two force-controlled arms, and two parallel jaw grippers with force sensors installed on the fingertips of each arm. Many well-known ROS packages, such as Moveit [56], Gazebo, and RViz, were first developed and tested on

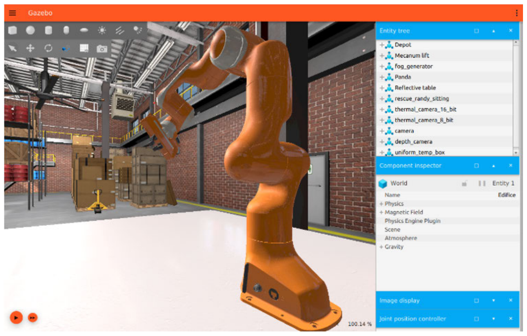


Figure 1.14: ROS Gazebo simulation environment [54]

the PR2 robot platform. Even though the PR2 robot has been discontinued, it is still a milestone of the ROS history and being used in worldwide research institutes and laboratories. Figure 1.15 shows the PR2 robot performing various tasks.

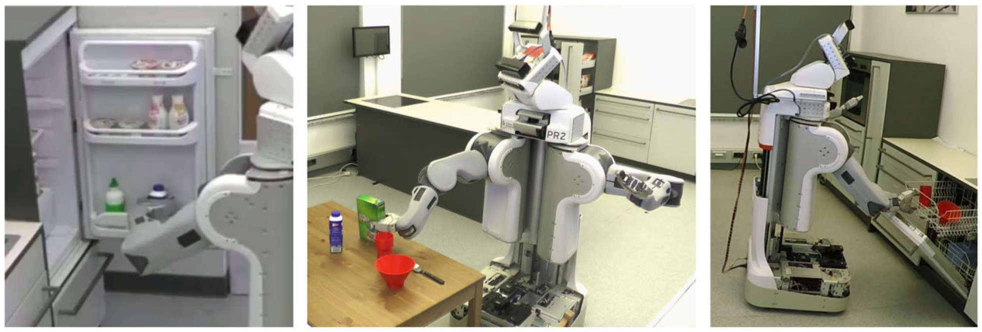
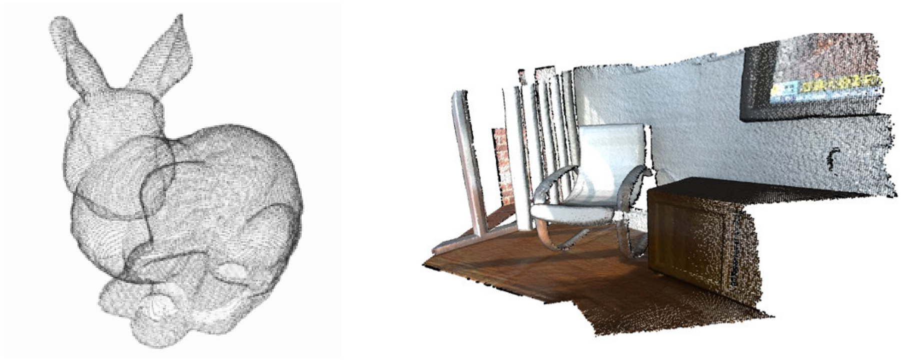


Figure 1.15: PR2 robot performing various tasks [57]

In addition to the traditional robot arms, ROS supports the development of multi-axis unmanned aerial vehicles(UAV) [58], underwater unmanned vehicles(UUV) [59], among others [60, 61, 62]. Many useful applications and functions including motion planning, controller interface, simulation platforms, autonomous navigation functions, machine vision functions can be found in the ROS function libraries. The software resources and open communication protocol promote the development work of researchers.

1.2.5 Overview of Point Cloud Technology

The point cloud data is a set of points in 3D space, each of which has its set of Cartesian coordinates (X, Y, Z). As shown in Fig. 1.16, point cloud data is usually captured by a 3D scanner or generated by 3D structure design software that represents the external surface information of objects around them [63]. Point cloud data is currently used in various fields, including reconstructing the 3D model of manufactured parts for surface quality inspection, indoor and outdoor mapping, human-computer interaction, virtual reality, autonomous driving and many other fields. When the point cloud technology was first introduced, the high cost of point cloud acquiring equipment severely hindered the development of point cloud technology.



(a) Point cloud of a bunny (b) Point cloud of an office

Figure 1.16: Point cloud example [64, 65]

As shown in Fig. 1.17, since 2010, Microsoft Kinect [66], Intel Realsense [67], Asus Xtion [68] and many consumer-grade low-cost point cloud acquisition devices

(RGB-D camera) have appeared on the market, whose prices are only 10% - 20% of those of the high-end laser scanners (Approx.10,000 - 30,000USD) [69]. In addition to the hardware like camera, the software and algorithm for point cloud data processing are also being developed. The most famous cross-platform open source libraries are PCL [70] and Open3D [71], which encapsulate common data structure classes and many general algorithms. Including point cloud data acquisition, visualization, filtering, denoising, segmentation, refinement, feature extraction and other algorithms. With the development of machine learning(ML) and reinforcement learning(RL), array of new applications will emerge in the future.

(a) Microsoft Kinect v1 (b) Microsoft Kinect v2

(c) Intel Realsense D435 (d) Asus Xtion

Figure 1.17: Consumer-grade RGB-D cameras [66, 67, 68]

In order to meet the increasing demands from industry to scan and deburr irregular and varied shapes, the robot deburring trajectory planning methods based on point cloud become a research focus in recent years [34], [35]. There are two main approaches for deburring trajectory planning based on point cloud data:

1. Directly using the point cloud data to generate robot deburring trajectory.

2. Using the point cloud data to reconstruct the 3D model of the object, and generating deburring trajectory based on the reconstructed 3D model.

For the first approach, researchers proposed a method to generate B-spline curves tool path using the raw point cloud data for three axis-milling machine, which was verified in Matlab [36]. The point cloud segmentation for robot deburring trajectory planning also has significant potential in manufacturing. For example, by combining the point cloud slicing and smoothing method, an optimization approach based on point cloud was developed to ensure that the grinding tool and work piece are in maximum contact rate [37]. In order to increase the efficiency for deburring tool path to capture a 3D shape, the point cloud data can be projected onto a 2D plane to plan the tool path, then the tool path is projected back to the 3D shape [38].

For the second approach. Based on the point cloud data captured with a high-resolution camera, researchers proposed a method to reconstruct the 3D model of parts, comparing the reconstructed 3D model with the reference CAD model in a computer to extract the area to be processed, and generating the manufacturing trajectory [72]. Another group of researchers migrated the method into a robot system to scan and generate the 3D model of steel parts and plan the robot surfacing trajectory [73].

Despite substantial studies on robot-based deburring trajectory planning, current state-of-the-art research mainly focuses on rigid objects with regular shapes, which eases the modelling process before deburring. Unfortunately, there is no common way to generate a robot deburring trajectory to process soft objects and objects with varied shapes, seafood processing being a typical challenge. In this paper, the Microsoft Azure Kinect RGB-D camera [66] is used to capture the 2D image and 3D point cloud data of the porcupine crab. Since the resolution of point cloud is limited, detailed features of the thin crab spines cannot be reconstructed well. Two novel methods are proposed to generate the robot trajectory for spine removal of the porcupine crab. The first one is based on the novel point cloud slicing and the nearest points searching method, the generated tool path covers the whole surface area of the crab shell and can remove all the spines [74]. It is found in the experiment that the accuracy and efficiency of the first method are relatively low. Hence, second method is proposed, combining the 2D image and the 3D point cloud data. Firstly, using the image processing method to locate the position of all the spines in the 2D image, and then using the ArUco code [75] as a bridge to establish the relation between the 2D image and 3D point cloud data. After that, the equation of the central lines of each spine can be calculated and used as input for spine removal tool path generation.

1.3 Contributions of the Thesis

The main objective of this research is to design, develop, simulate and test the robot-based porcupine crab spine removal system.

(1) The primary contribution of this thesis is developing a robot-based porcupine crab spine removal working station, including proposal design, mechanical and electrical parts selection, diagram design and drawing, programming and simulation, part purchasing, system installation and commissioning, testing and modification.

(2) Using the data from the 3D RGB-D camera, the thesis demonstrates two

novel crab spine identification and processing approaches, providing a novel solution to accomplish high accuracy recognition using a less accurate camera.

(3) This thesis also validates the method in a simulation environment and real world with a real robot arm and a 3D-printed porcupine crab model. The result shows that the thesis achieved all expected results.

1.4 Summary

This chapter has presented the overview introduction and the overall structure of this thesis, and provided a review of the porcupine crab, the state-of-art seafood processing systems, robot-based deburring systems, the robots and robot operating system (ROS), and the point cloud technologies. The reviewed works provides some information about the cutting-edge research and also gives inspiration for the thesis.

Chapter 2

Overall System Desgin

2.1 Introduction

In this chapter, the overall design of the robot-based porcupine crab spine removal system is introduced. Firstly, the design principles and requirements of the whole system are presented. In addition, the selection principle and the final selection results for all major components of the system, including the robot arm, the controller, the working platform and the camera, are provided. Afterward, the overall electrical and control system design and the network connection layout design are proposed. Moreover, the detailed design of the mechanical system, electrical system and simulation environment are also discussed in this chapter.

2.2 Design Philosophy

As introduced in Chapter 1, the robot-based porcupine crab spine removal system will be used to identify the crab and remove all the spines automatically. Design philosophy for the system are summarized below:

1. **Safety.** There are two main safety requirements that must be considered carefully. The first one is the robot's safety for humans, not only in the laboratory environment but also in the potential production environment where the equipment will be used in the future, the robot system will share the working space with human workers and operators. According to the EN/ISO 10218 (robots and robotic devices: safety requirements for industrial robots) [46], robotic systems are required to be safe for humans in the vicinity during operation and be stopped immediately when something abnormal happens. Secondly, the robot system also needs specific safety requirements, including preventing the robot from colliding with other parts and stopping the system when some key parameters exceed the allowed limit.

2. **Flexibility.** The design of the system needs to be highly flexible and customizable. The system design may constantly be revised and changed during the project development process according to the experimental results. At the same time, the robot system may be used for the development of different projects in parallel, which requires the whole system to be flexible for rapid modification.

3. **Low-cost.** The development of new technology often faces much uncertainty and requires constant experiment and optimization, making the cost a major consideration. Due to the high cost and the long manufacturing cycle of the traditional machining method, the 3D printing technology is mainly used to produce the experimental prototype. Meanwhile, for the crab data acquisition, a low-cost 3D RGB-D camera is used in this project, the innovative algorithm enables the low-cost camera to achieve high-precision recognition result.

2.3 Overall System Design

The robot-based spine removal application is complex and can be affected by many factors. Crabs differ from body sizes and spines locations, which places high demands on the system design. As shown in Fig. 2.1, this system consists of a robot with its control box, an end-effector that integrates with a 3D RGB-D camera and a spine removal cutting tool, a central control workstation and an aluminum extrusion working platform. More information about each part of the system will be introduced in the following sections.

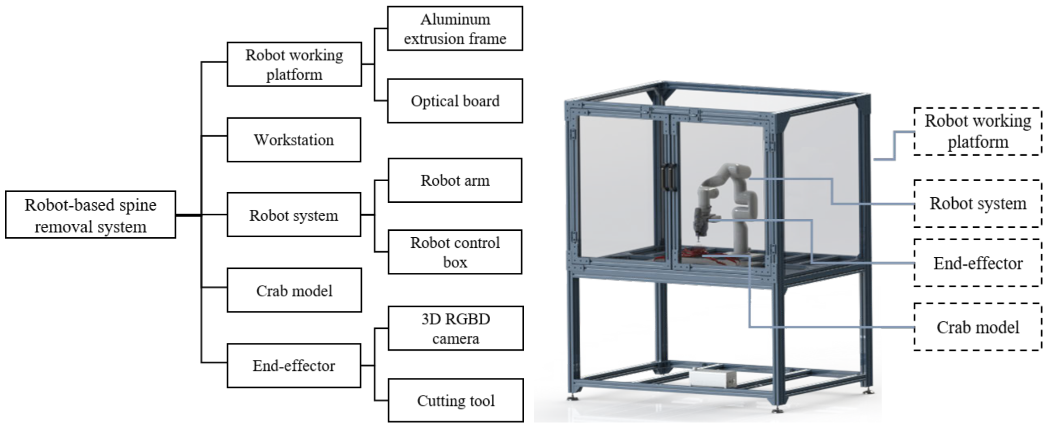


Figure 2.1: Structure of the Robot-based spine removal system

2.3.1 Robot System Selection

As the core equipment of the robot-based spine removal system, the robot will be used to carry the camera for crab 2D image and 3D point cloud data acquisition and also carry the cutting tool for removing the crab's spines. Therefore, the selection of the robot is one crucial step for building the whole system. The main parameters to be considered include the robot type, load capacity, working range, safety, expandability. Since the spine removal operation requires the robot to constantly change its posture, according to Section 1.2.4.1, the parallel robot has limited ability to perform continuous complex movements, and have low degrees of freedom. A serial robot is selected instead of its parallel counterpart.

Currently, there are two main types of serial robots, the traditional industrial and collaborative robots(Cobot), the main difference of which lying in the parameter values, as compared in Table 2.1.

Table 2.1: Industrial robot and collaborative robot comparison

	Industrial Robot	Collaborative Robot	Project requirement
Payload	Up to 2300 kg	Up to 35 kg	5 kg
Repeatability	(+/−) 0.02–0.1 mm	(+/−) 0.1–0.2 mm	(+/−) 0.2 mm
Working range	Up to 4900 mm	Up to 1900 mm	With in 1000 mm
Power supply	200-575V AC	110 -240 V AC	120V AC
Body volume	Big batches	Low-volume	Low volume
Deployment	Complex	Fast and Easy	Fast
Safety	Not safe	Collaborative and safe	Safe
Investment	High	Low	Low

The power supply in the laboratory is 120V AC. Based on the size of the crab, the robot's working range should exceed 0.6 m, the total weight of the end-effector with the 3D RGB-D camera and cutting tool is 2.6 kg, the estimated cutting force obtained from manual spine removal experiments is about 20N, which requires the payload of the robot to be higher than 5 kg. The robot needs to have a safety function and be able to stop running immediately to prevent causing injury to humans and other equipment when a collision occurs. Also, the robot control system must be extensible, such as supporting the multiple digital and analog signal input and output, different communication protocols, etc.

Based on the selection principles, the xArm6 collaborative robot and its controller are selected in this thesis. The xArm6 robot shown in Fig. 2.2 has a payload of 5 kg, a working range of 700 mm, and a repeatability of $+/-$ 0.1 mm, the robot also supports Modbus and Ethernet communication protocols and is also equipped with a safety function. The overall parameters of the robot can meet the requirement of this thesis.

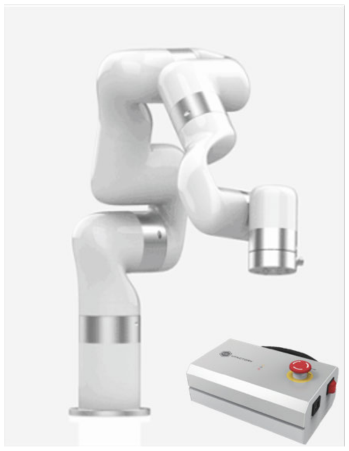


Figure 2.2: xArm6 robot and its control box

2.3.2 3D RGB-D Camera Selection

As mentioned in Section 1.2.5, Microsoft Azure Kinect camera [66], the most famous low-cost camera, is used in this thesis, the main applications including 3D scene reconstruction, human body tracking, autonomous driving, virtual reality and augmented reality applications. As shown in Fig. 2.3, it contains a high-resolution 2D RGB sensor, a 3D depth sensor (TOF)[1], a microphone array and a motion sensor (IMU), which enables the camera to capture the 2D RGB images, 3D depth images, camera posture and surrounding sound in the environment.

In this project, the camera is used to capture the 2D image and 3D depth information. The output resolution of the 2D camera is 3840×2160 (5-30 FPS), with various image formats such as RGBA, YUV, YUY2, and NV12. The 3D depth sensor has a resolution of 1024×1024 (5-30 FPS), with a measurement range of 0.25 – 3.86 m, each pixel in the depth image indicates the XYZ coordinates of the spatial point. The internal firmware of the camera can convert the 3D depth image into the .PCD and .PLY format of the point cloud data. This camera can be connected to the workstation via USB-C cable, and Microsoft provides the Python and C++ APIs for easy integration and development.

2.3.3 Workstation Selection

In this thesis, the 2D image and 3D point cloud data are needed to be processed in parallel with the robot control in real-time. Meanwhile, the entire experiment needs to be visualized, which places a high requirement on the computer hardware.

To meet the requirements of the experiment, a high-performance workstation is chosen as the central control unit. As shown in Fig. 2.4, the model is Dell-XPS-

[1]TOF: A time-of-flight camera (ToF camera) is a ranging imaging camera system that uses time-of-flight technology to resolve the distance between the camera and the subject at each point in the image by measuring the round-trip time of an artificial light signal provided by a laser or LED.

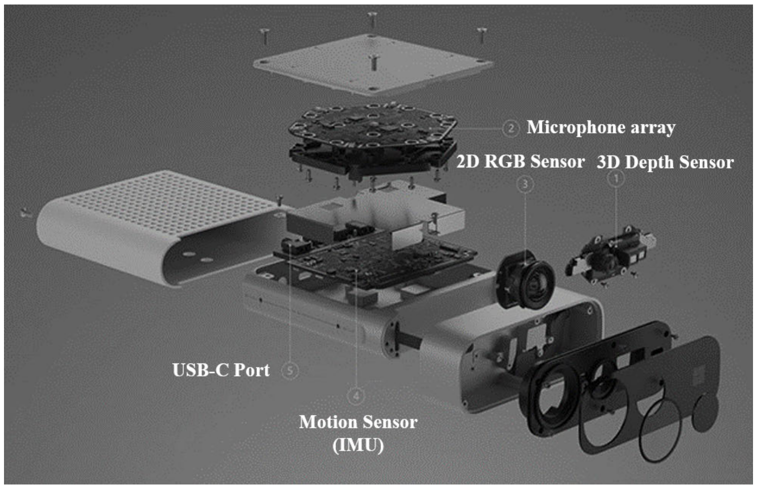


Figure 2.3: Microsoft Azure Kinect Camera Structure [66]

8940 [76] with a 3.8GHz Intel Core i7 10700K CPU with 8 cores and 16 threads, each thread can be assigned to process different tasks parallelly, increasing the multi-task processing performance. 32GB DDR4 RAM is used in the workstation, allowing the robot control system, image processing system and visualization system to run simultaneously. Nvidia RTX 3070 graphic processing unit is used, the 8G memory and 5888 CUDA cores[2] can significantly accelerate the processing speed of point cloud data.

Regarding the overall performance of the control system, the camera and robot require high accuracy for computational solutions, which calls for a system with a high data transmission speed in addition to the basic requirements. Ubuntu 18.04 LTS, a Linux-based operating system with high system throughput and time-sharing characteristics, is chosen based on which the ROS system is built for real-time parameters acquisition and control of the system.

[2]CUDA cores: Nvidia GPUs host hundreds or thousands of cuda cores. These cores are responsible for processing all the input to and output data of the GPU, performing calculations.

Figure 2.4: Dell XPS 8940 Workstation [76]

2.3.4 Cutting Tool Selection

The cutting tool is mounted on the robot, which is carried by the robot to cut the spines on the surface of crab's body. When making the selection, the following requirements are taken into consideration. Firstly, the size and weight of the cutting tool need to be as small as possible to optimize the robot's load capacity and ensure that the robot's working range is not affected. Secondly, the blades of the cutting tools need to be interchangeable, so that they can be easily replaced during the experiment. Furthermore, the price of the cutting tool also needs to be considered.

As shown in Fig. 2.5, the Dremel 3000 rotary tool [77] with reinforced fibreglass cutting disc is chosen as the cutting tool. This tool is widely used in wood carving, metal polishing, among others. The weight is 0.62 kg and a speed range of 5,000 – 32,000 rpm. It has a removable threaded cap that can easily and preciously be installed into the end-effector with high rigidity and precision.

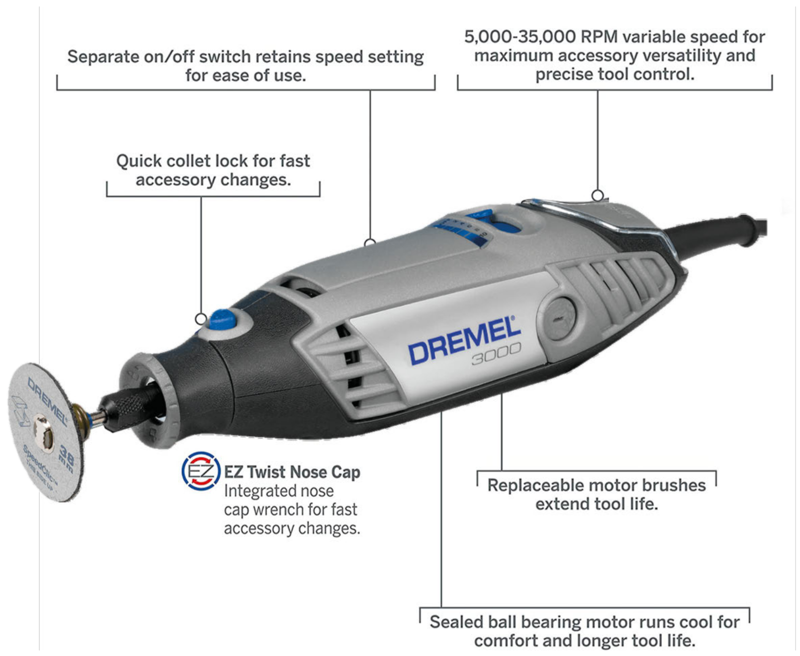

Separate on/off switch retains speed setting for ease of use.
5,000-35,000 RPM variable speed for maximum accessory versatility and precise tool control.
Quick collet lock for fast accessory changes.
DREMEL 3000
EZ Twist Nose Cap Integrated nose cap wrench for fast accessory changes.
Replaceable motor brushes extend tool life.
Sealed ball bearing motor runs cool for comfort and longer tool life.

Figure 2.5: Dremel 3000 rotary tool [77]

2.4 Mechanical Design

2.4.1 Robot Working Platform Design

In this thesis, aluminum extrusion part is selected to build the robot's working platform. In comparison to steel, aluminum extrusion is a lightweight, high-strength, and customizable material that is widely utilized in automation production lines and research laboratories. The following requirements are mainly considered during the design phase:

1. The robot should have enough working range on the platform.

2. Although the robot has a collision-safe function, some swarf may splash out during the cutting process, so the working platform needs to be isolated.

3. Experiment objects should be easily installed and exchanged on the platform.

4. The platform needs to be highly expandable to perform various experiments with a few modifications.

2.4.1.1 Robot Working Range Analysis

The robot working space represents the set of points in space that the end of the robot arm can reach. Researchers have proposed graphical and analytical methods to calculate the robot working range [78, 79]. The graphical method is by manually dragging the end effector of the robot in a simulation environment to find the robot working range, which is relatively simple but has low precision. The analytical method is by iterating through the angles of the robot's axes and calculating all possible robot poses using forward kinematic algorithm. Although this method has high precision but less of efficient. This thesis proposes a numerical method to solve the problem, the Monte Carlo method [80], which mainly uses random input parameters to predict the probability of various outcomes. This method can calculate and visualize the robot's working range intuitively and quickly, thus being suitable for any articulated robot.

The Monte Carlo-based robot working range calculation method is as follows: generating a series of random joint angles of the robot and calculating the robot endpoint coordinate in the reference system using the forward kinematic method [81].

Figure 2.6 shows the flow chart of the Monte Carlo-based robot working range calculation method, the first step is to load the robot Denavit-Hartenberg (DH) model[3].

[3]Denavit-Hartenberg(DH): DH parameters [81] were introduced by Jacques Denavit and Richard S.Hartenberg. It is used in the robotics research field to describe the structure and properties of robot arms, including joint axis orientations, link length and other parameters.

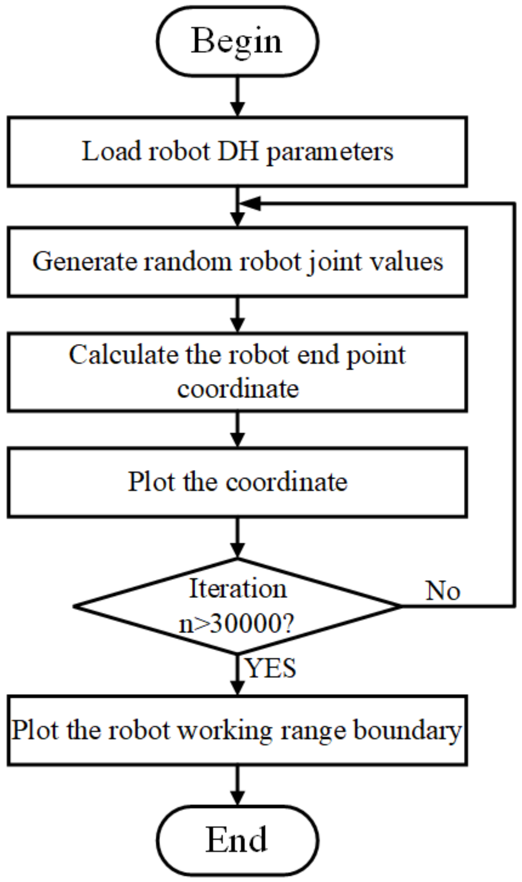

Begin
Load robot DH parameters
Generate random robot joint values
Calculate the robot end point coordinate
Plot the coordinate
Iteration n>30000?
No
YES
Plot the robot working range boundary
End

Figure 2.6: Flow chart of Monte Carlo-based robot working range calculation method

The standard DH parameters of the robot are shown in Fig. 2.7 and Table 2.2. Four parameters are used to describe the structure of the robot. θ_i is called the *joint angle*, which represents the rotation angle about the Z_{i-1} axis, from X_{i-1} to X_i by angle θ_i. d_i is called the *link offset* distance, which represents the offset distance d_i along Z_{i-1} to the common normal. a is called the *link length*, which represents the length of the common normal. α_i is called the *link twist* angle, which represents the rotation angle from Z_{i-1} axis to the Z_i axis. *Offset* is the offset joint angle from the mathematical zero position to the mechanical zero position as shown in Fig. 2.7.

After loading the robot DH model, the next step is to calculate the robot end point

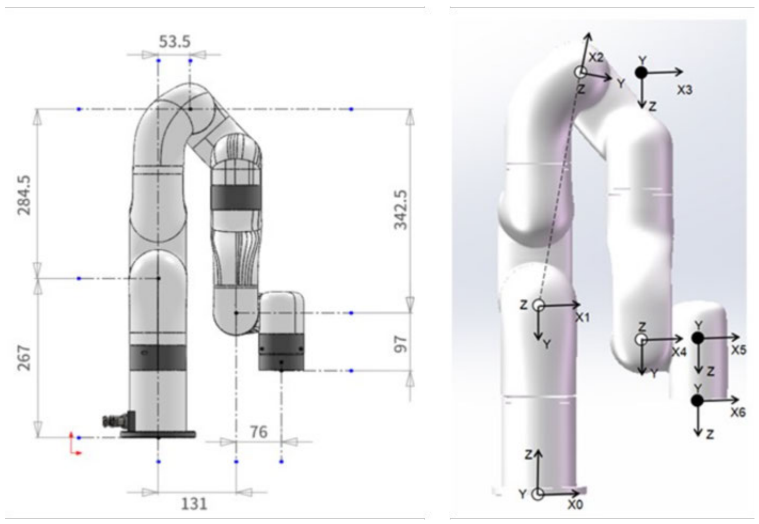


Figure 2.7: xArm6 robot DH parameters

Table 2.2: xArm6 robot DH parameters

Kinematics	θ**(rad)**	d**(mm)**	α**(rad)**	a**(mm)**
Joint 1	θ_1	267	$-\pi/2$	0
Joint 2	θ_2	0	0	289.49
Joint 3	θ_3	0	$-\pi/2$	77.5
Joint 4	θ_4	342.5	$\pi/2$	0
Joint 5	θ_5	0	$-\pi/2$	76
Joint 6	θ_6	97	0	0

coordinates based on each joint angle. In this thesis, the Forward Kinematics (FK) method is used, eq. (2.1) shows the general form of the homogeneous transformation matrix(HTM) to calculate the relation of each coordinate frame to the next one in the robot, where $\mathbf{R}_i^{i-1}$ is the rotation matrix from the frame i to frame $i-1$, $\mathbf{p}_i^{i-1}$ being the vector pointing from the origin of frame i to that of frame $i-1$.

$$\mathbf{T}_i^{i-1} = \left[\begin{array}{ccc|c} & \mathbf{R}_i^{i-1} & & \mathbf{p}_i^{i-1} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \left[\begin{array}{ccc|c} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{2.1}$$

As shown in eq. (2.2), the xArm6 robot end point position can be calculated by multiplying all the HTMs of the robot together, starting from the first joint to the last one. Substituting the DH parameters into eq. (2.2), all the HTMs of the robot are obtained, as demonstrated from eq. (2.3) to eq. (2.5), where $S_i = \sin\theta_i$, $C_i = \cos\theta_i$, θ_i is the joint angle, i means the joint number of the robot.

$$\mathbf{T}_6^0 = \left[\begin{array}{ccc|c} & \mathbf{R}_6^0 & & \mathbf{p}_6^0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = [\mathbf{T}_1^0\mathbf{T}_2^1\mathbf{T}_3^2\mathbf{T}_4^3\mathbf{T}_5^4\mathbf{T}_6^5] \tag{2.2}$$

$$\mathbf{T}_1^0 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & 267 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{T}_2^1 = \begin{bmatrix} C_2 & -S_2 & 0 & 289.49C_2 \\ S_2 & C_2 & 0 & 289.49S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$\mathbf{T}_3^2 = \begin{bmatrix} C_3 & 0 & -S_3 & 77.5C_3 \\ S_3 & 0 & C_3 & 77.5S_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{T}_4^3 = \begin{bmatrix} C_4 & 0 & S_4 & 0 \\ S_4 & 0 & -C_4 & 0 \\ 0 & 1 & 0 & 342.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$\mathbf{T}_5^4 = \begin{bmatrix} C_5 & 0 & -S_5 & 76C_5 \\ S_5 & 0 & C_5 & 76S_5 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{T}_6^5 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & 97 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

As the position of the robot end point in space is the only factor in calculating the robot working range, the kinematics equation of the robot end point are shown in eqs. (2.6) – (2.9).

$$\mathbf{T}_6^0 = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{2.6}$$

$$\begin{aligned} p_x = &-78C_1S_3S_2 + 76S_5\left(-C_1C_2S_3 - C_3C_1S_2\right) + 342.5\left(-C_1C_2S_3 - C_3C_1S_2\right) \\ &+ 76C_5\left(C_4\left(C_3C_1C_2 - C_1S_3S_2\right) + S_4S_1\right) + 97\left(C_5\left(-C_1C_2S_3 - C_3C_1S_2\right)\right. \\ &\left.- S_5\left(C_4\left(C_3C_1C_2 - C_1S_3S_2\right) + S_4S_1\right)\right) + 78C_3C_1C_2 + 289C_1C_2 \end{aligned} \tag{2.7}$$

$$\begin{aligned} p_y = &\,78C_3C_2S_1 + 289C_2S_1 + 76S_5\left(-C_2S_3S_1 - C_3S_2S_1\right) + 342.5\left(-C_2S_3S_1 - C_3S_2S_1\right) \\ &+ 76C_5\left(C_4\left(C_3C_2S_1 - S_3S_1S_2\right) - C_1S_4\right) + 97\left(C_5\left(-C_2S_3S_1 - C_3S_2S_1\right)\right. \\ &\left.- S_5\left(C_4\left(C_3C_2S_1 - S_3S_1S_2\right) - C_2S_4\right)\right) - 78S_3S_2S_1 \end{aligned} \tag{2.8}$$

$$\begin{aligned} p_z = &-78C_2S_3 - 78C_3S_2 + 76C_4C_5\left(-C_2S_3 - C_3S_2\right) + 76S_5\left(S_3S_2 - C_3C_2\right) \\ &+ 342.5\left(S_3S_2 - C_3C_2\right) + 97\left(C_5\left(S_3S_2 - C_3C_2\right) - C_4S_5\left(-C_2S_3 - C_3S_2\right)\right) - 289S_2 + 267 \end{aligned} \tag{2.9}$$

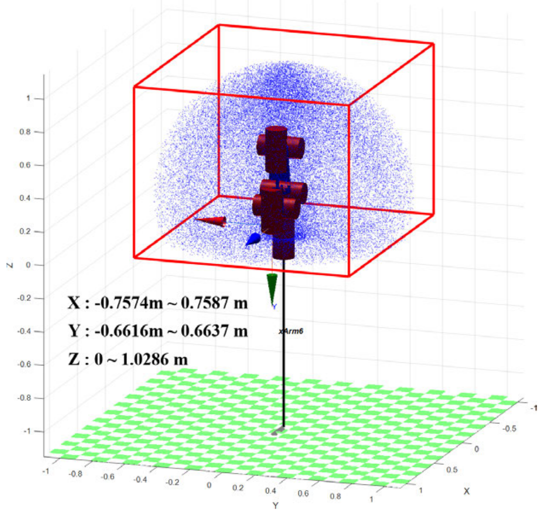
As shown in eq. (2.10), each joint angle is randomly generated within its lower and upper limits using the uniformly distributed random numbers. The $\theta_i^{\min}$ and $\theta_i^{\max}$ denote the working range of joint i, RAND() being a function for generating uniformly distributed random numbers [82].

$$\theta_i = \left(\theta_i^{\max} - \theta_i^{\min}\right) \times \mathrm{RAND}() \tag{2.10}$$

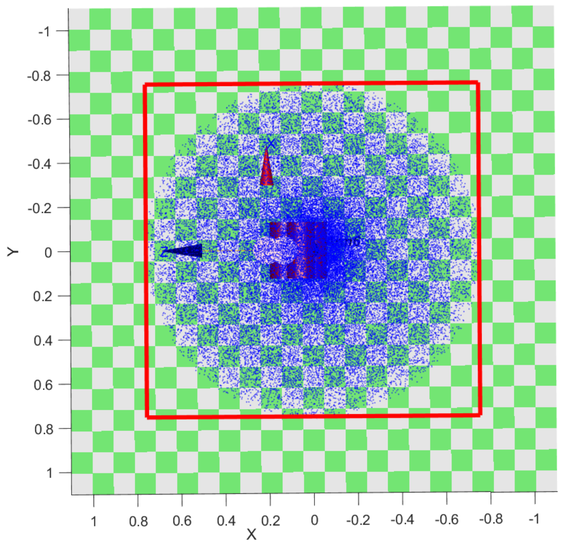
30,000 sets of robot joint angles have been generated using the iteration method. By substituting these random data into eq. (2.6), a set of robot end point coordinates in the robot working space can be calculated. Fig. 2.8 shows the simulation results of the robot working range in Matlab, including the projections of the workspace on XOY, XOZ and YOZ planes, respectively. It can be seen that these endpoints are uniformly distributed, which can straightforwardly describe the robot's working range, namely, $X \in [-0.757$ m, 0.758 m$]$, $Y \in [-0.661$ m, 0.664 m$]$ and $Z \in [0$ m, 1.029 m$]$.

2.4.1.2 Robot Working Platform Structure Design

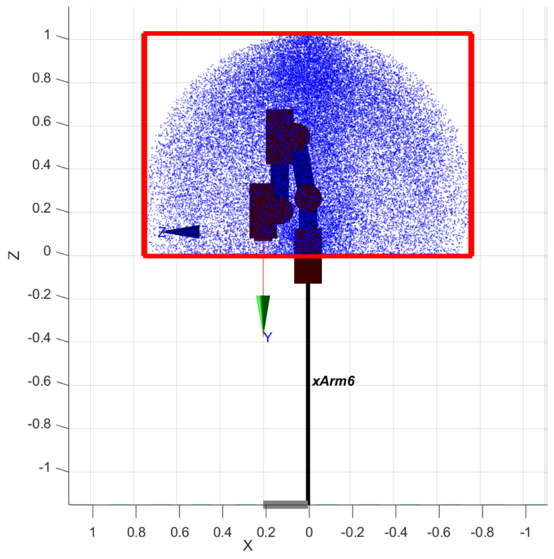
The robot working platform structure is designed according to the design requirement and the calculated robot working range in the previous sections. As shown in Fig. 2.9(a), the main structure of the platform is made of aluminum extrusion, which has the advantages of high strength and being customizable. This platform is divided into two layers – the lower and upper layers. The robot control box, network switch and power strip are placed on the lower layer while the robot arm and the aluminum optical breadboard that is used to fix the research object, are installed on the upper layer. To ensure safety during the experiment, the upper layer is isolated using



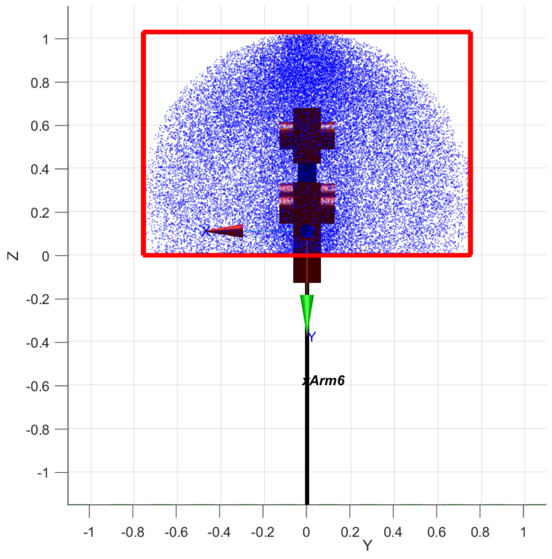
(a) Robot working range



(b) Robot working range in the XOY projection



(c) Robot working range in the XOZ projection

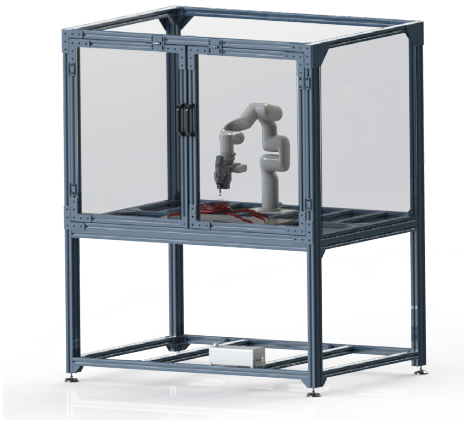


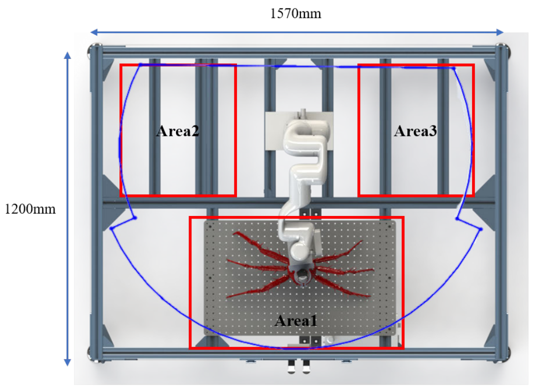
(d) Robot working range in the YOZ projection

Figure 2.8: xArm6 robot working range analysis based on Monte Carlo method

transparent acrylic panels and the structure is also grounded. A pair of doors is set in front of the robot to facilitate workpiece installation and equipment maintenance.

Figure 2.9(b) shows the top view of the robot workstation, in which the blue curve


(a)



(b)

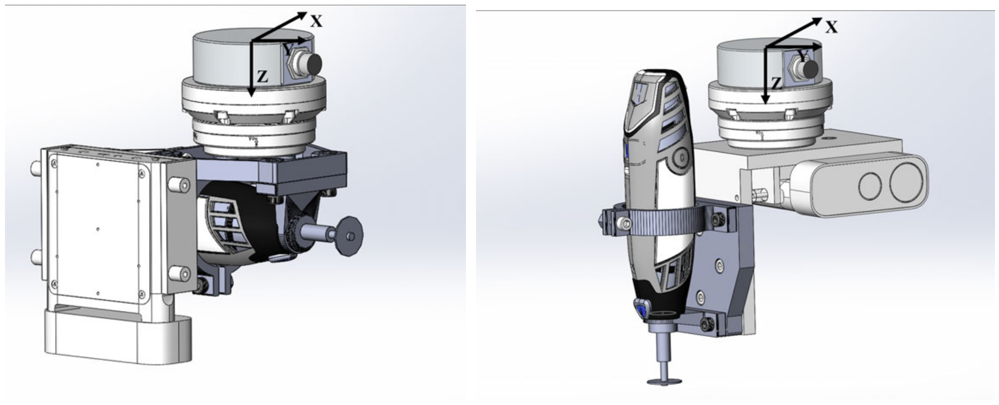
Figure 2.9: Robot working platform: (a) Isometric view (b) Top view

represents the robot working range. In order to extend the robot operating range and let the robot be able to perform different experiments at the same, three working areas (Areas 1, 2, and 3) have been set up. An aluminum optical breadboard made of solid aluminum alloy with a series of threaded holes is installed in Area 1 to fasten the crab model and perform the experiment for spine removal, which is convenient for fixing equipment and parts on the top of the board. Support structures are also provided in Areas 2 and 3, where other experimental equipment can be installed. Also, this robot working platform is capable of meeting the requirement of the EN/ISO 10218 safety standard [46].

2.4.2 Robot End-effector Design

To increase the integration level of the system and let the robot perform various tasks without disassembling the end-effector, this thesis proposes a method to integrate the cutting tool and the 3D camera into the end-effector. After acquisition of the

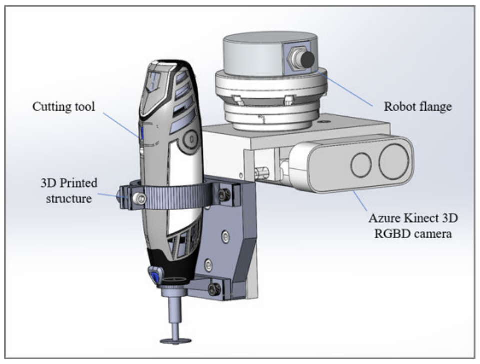
2D image and 3D point cloud data, the robot can instantly start the spine removal operation. Also, the relative locations of the robot flange, the camera and the end-effector are fixed, which makes the system only need to be calibrated once. Figure 2.10 shows the first and second versions of the end-effector design. In the first version, the cutting tool is mounted along the Y-axis, and the camera is mounted along the Z-axis of the robot flange coordinate system. This design is compact, but during the simulation, it is found that the robot would reach the wrist singularity points [83] during the spine removal process, which will stop the motion of the robot or cause a discontinuous trajectory. Then, the second design was proposed, in which the mounting direction of the cutting tool and that of the camera are swapped. Thus the singularity problem is solved and verified in the simulation environment.



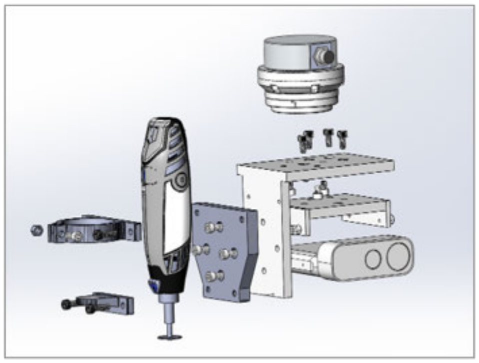
(a) First version (b) Second version

Figure 2.10: Robot end-effector design

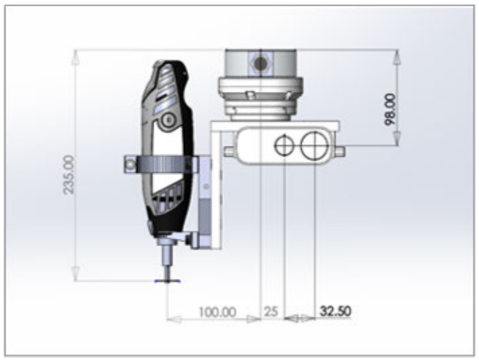
Figure 2.11 shows the detailed design of the robot end-effector, which is consisted the cutting tool, the 3D camera and the 3D printed installation structure. In order to reduce the calibration difficulty of the camera and the cutting tool, and also decrease the calculation difficulty of the robot trajectory, the distance between the camera lens and the endpoint of the cutting tool with the center of the robot flange is designed to be an integer. The main structure of the end-effector is 3D–printed with PETG material [84] which has high tenacity and strength. Parts are connected via bolts and nuts to ensure a high connection strength and precision.



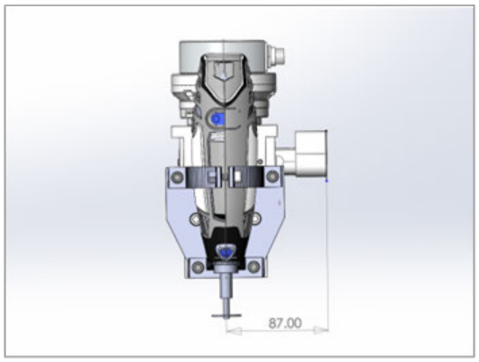
(a) Isometric view



(b) Exploded view



(c) Front view



(d) Side view

Figure 2.11: Robot end-effector design

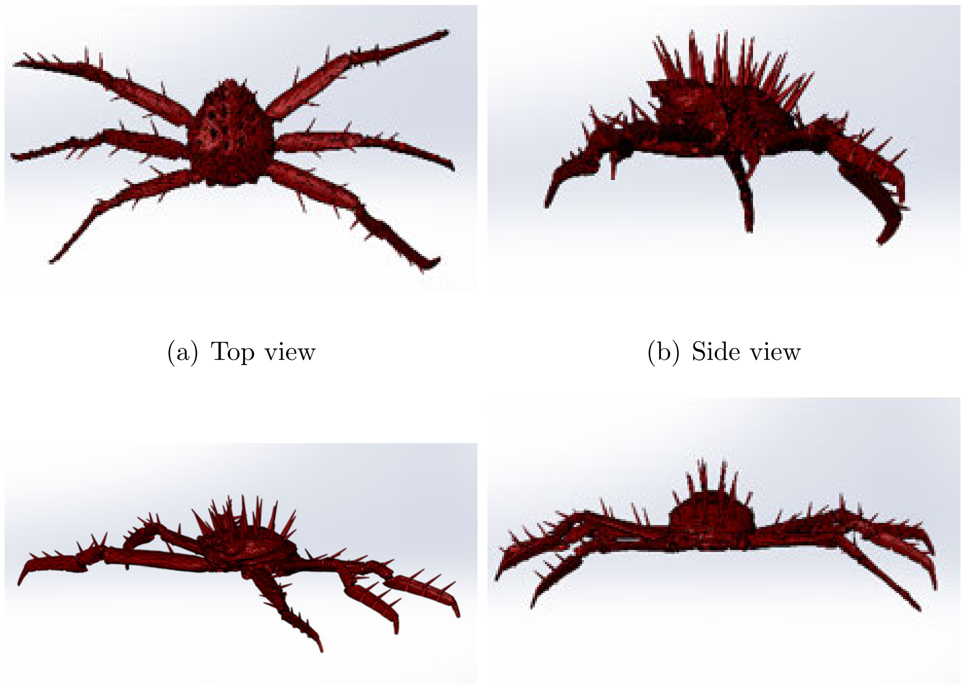
2.4.3 Porcupine Crab 3D Model Design

Limited by the quantities and the storage difficulty of the crab, it is not feasible to use real crabs to do the experiment. In this thesis, based on the size and feature information of the real porcupine crab, an accurate 3D model of the crab is designed in the 3D modelling software and printed by a 3D printer.

In order to validate the effectiveness of the proposed method, the model dimension, the growth location and direction of the spines, the shape and size of the crab and other characteristics need to be consistent with the real porcupine crab as much as possible. Multiple crab models, which are easily manufactured and replaced at a low cost, are needed for repeated tests, Fig. 2.12 shows the 3D model design of the porcupine crab.

The complex shape and spine features of the crab make it nearly impossible to draw a complete and accurate model by hand, for the same reason, existing 3D scanners in the university could not be used to well-scan and reconstruct the crab's 3D model. Hence, there is no existing high-resolution 3D model of the porcupine crab. According to the previous chapters, the porcupine crab is a member of the king crab species with almost the same dimensions, surface and leg features as the current mainstream king crab species.

In order to keep the features of the crab as accurate as possible, the design of the porcupine crab model in this thesis is based on a high-resolution 3D model of a real king crab. Based on the measurement result of the real porcupine crab, the dimensions are adjusted, and the features of the porcupine crab's spines are added to the king crab model. Using the mean size of the porcupine crab as a reference [4], Fig. 2.13 shows the designed porcupine crab model with dimensions.



(a) Top view (b) Side view

(c) Isometric view (d) Rear view

Figure 2.12: Porcupine crab 3D model design

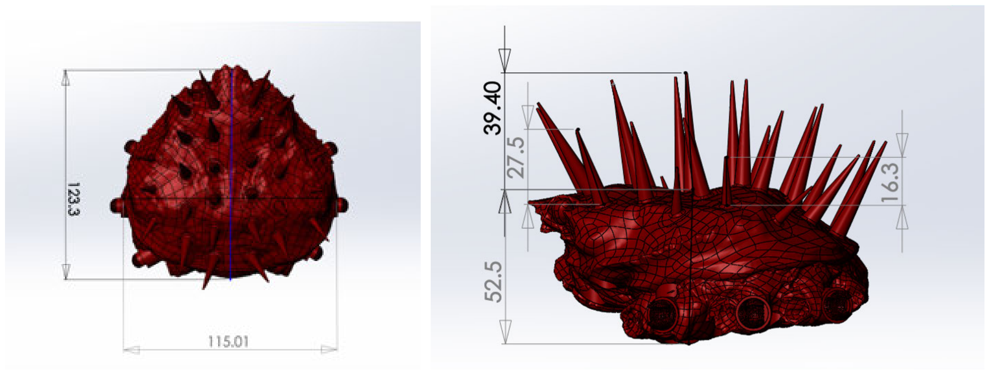
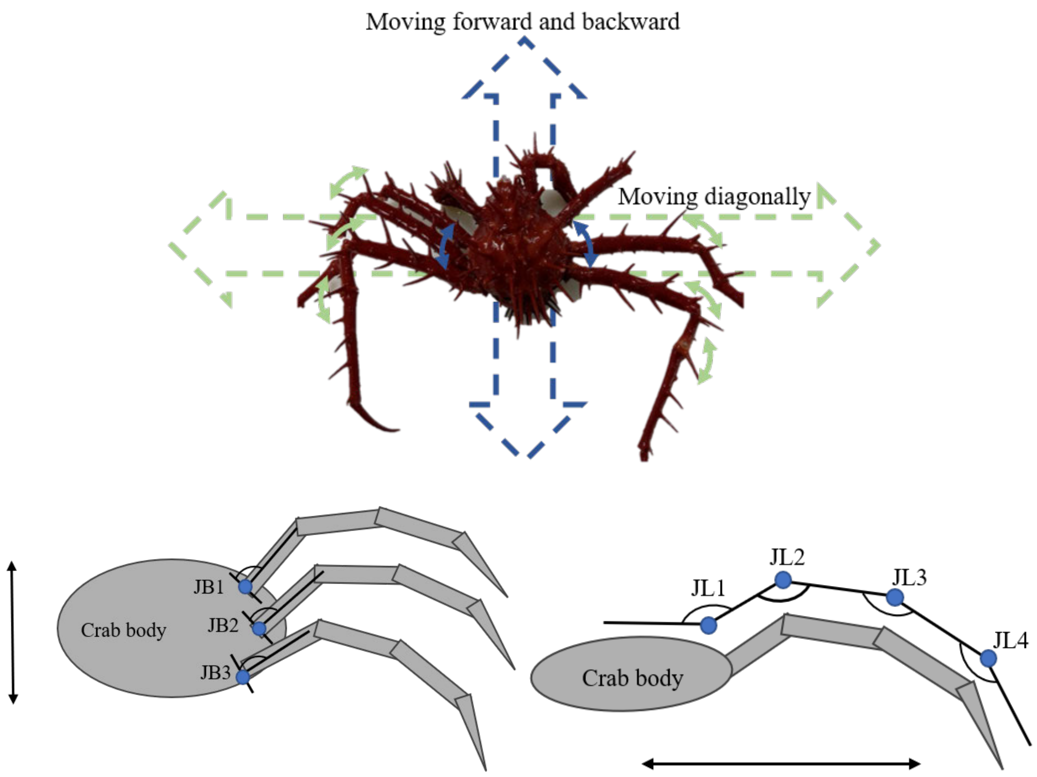


Figure 2.13: Porcupine crab 3D model

As shown in Fig. 2.14, in natural environment, the porcupine crab is able to move forward and backward with the movement of its body joints (*JB1-JB3*). Also, the crab's legs are able to rotate along its leg joints(*JL1-JL4*), giving the crab the ability to move diagonally [5].



(b) Moving forward and backward

(c) Moving diagonally

Figure 2.14: Porcupine crab movement

The crab model is designed to have similar motion characteristics as the real crab. Using the joint of the real porcupine crab as a reference, this thesis proposes a design of a ball joint with a special structure. As shown in Fig. 2.15, the ball joint consists of two parts and has two rotational degrees of freedom, which can well-simulate the motion of the crab's joint, and it is also suitable to be built by a 3D printer. The main research object of the thesis is the crab shell.

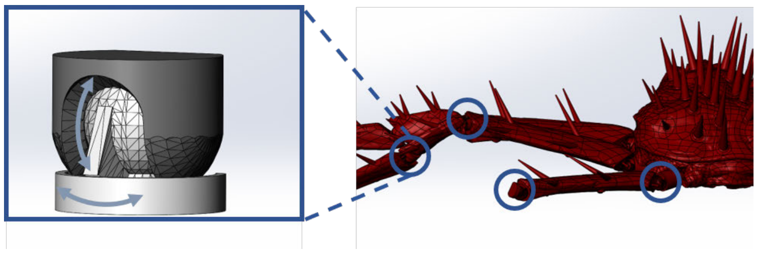


Figure 2.15: Ball joint structure

Multiple 3D printed crab models need to be prepared as experimental objects for iterative optimization and adjustment of the algorithm. In order to minimize the preparation time between experiments and reduce the waste of 3D printing materials (time and cost optimization), inspired by the charging plug of electric cars, this thesis proposes a novel design of crab spine part with a replaceable and detachable structure, the bottom end of the spine part is designed as a pin connector, which has a small bevel structure that makes it easier to insert and less likely to break. Multiple slots are set on the corresponding positions of the crab body, where the spine parts can be easily, securely and accurately fixed. Figure 2.16 shows the assembly of the porcupine crab model.

According to the characteristic of the real porcupine crab, spines are evenly distributed on the shell surface of the crab. They are growing in the direction perpendicular to the crab shell surface where the root of the spine is located. The shape of the spines is conical, and the spines in the middle region of the shell are longer in length and thicker in diameter than the spines near the edge [4]. The spine parts in this thesis are designed to mimic the location and dimensional characteristics of the

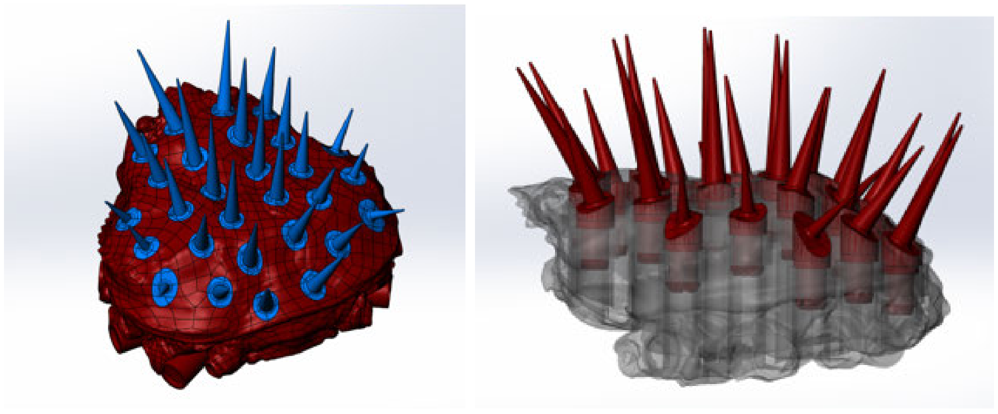


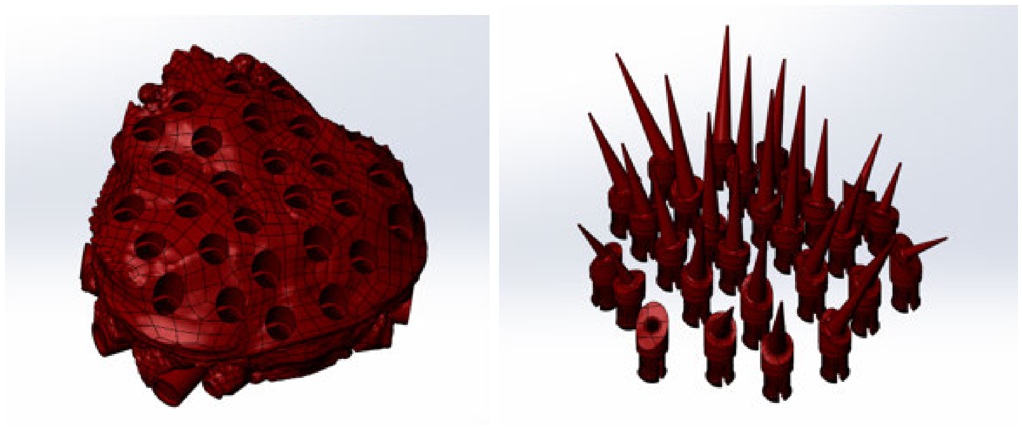
Figure 2.16: Assembly of the porcupine crab model

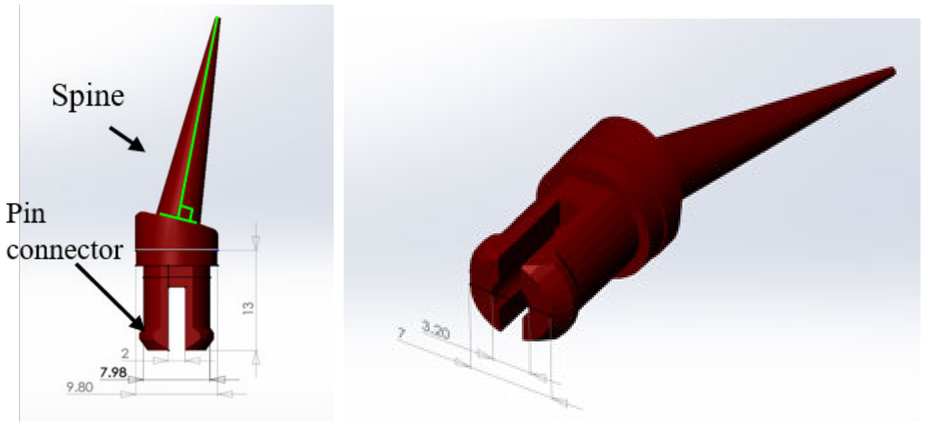
real porcupine crab, as shown in Fig. 2.17.

In this thesis, the crab model is manufactured by a 3D printer with the PETG [84] 3D printing material. The Shore hardness of the PETG material is 71.4D [85], similar that of the real porcupine crab shell, which makes it suitable to be used as the experiment material. The 3D–printed porcupine crab model is shown in Fig. 2.18.

2.5 Electrical System Design

Figure 2.19 shows the power supply and control system diagram of the robot-based spine removal system. In this system, the workstation is the central control unit, connected to the robot control box through a network switch via Ethernet TCP/IP protocol. The Microsoft Azure Kinect camera is connected to the workstation through a USB-C cable for transmitting images and point cloud data. The xArm6 robot arm is connected to the robot control box via a cable that can simultaneously transfer the control signal and provide the power supply. Two sets of power supply power the system: the workstation, robot control box and cutting tool are powered by 120 volts


(a) Crab shell part (b) Crab spine parts



(c) Spine part dimension 1 (d) Spine part dimension 2

Figure 2.17: Shell and spine parts of the porcupine crab model

AC, the network switch and the Microsoft Azure Kinect camera are powered by 12 volts DC. In order to ensure safety during the experiment and reduce the interference signal that is generated by the cutting tool when operating at a high rotation speed, all parts of the systems are well-grounded.

Redundancy design theory has been used in the system's development phase, leaving room for upgrading and optimization of the system. The system power supply has about 40% redundancy, 16 digital and 8 analog input and output ports are reserved

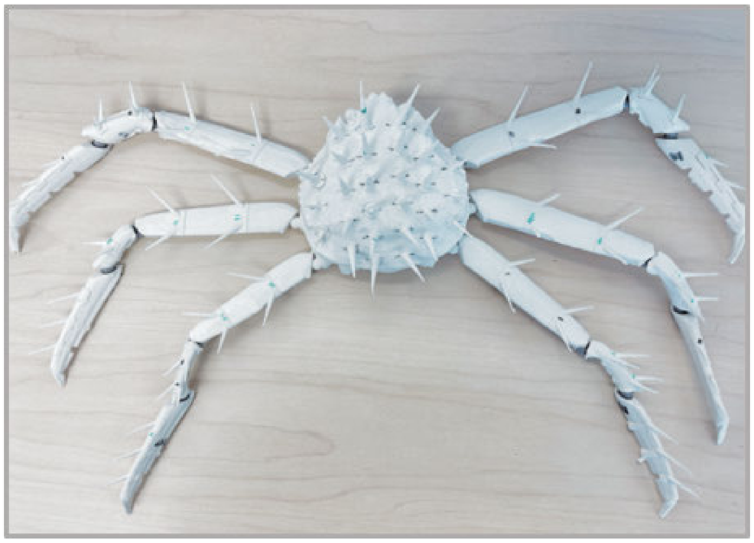


Figure 2.18: 3D-printed porcupine crab model

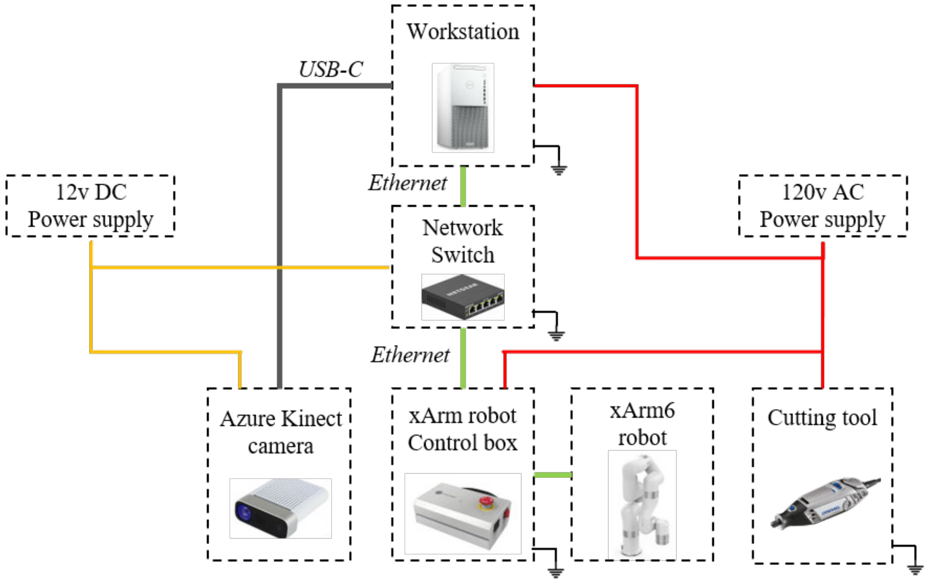


Figure 2.19: Robot-based spine removal system control and power diagram

that can be used to connect to relays, valves and other actuators, as well as digital and analog sensors for data acquisition.

2.6 Simulation System Setup

The thesis uses ROS as the simulation platform in which the simulated robot working station can be set as the real one in the real world. The 3D RGB-D camera and the robot with the same specifications as its real counterparts can be simulated, and the 3D model of the porcupine crab and robot end-effector can also be loaded into it.

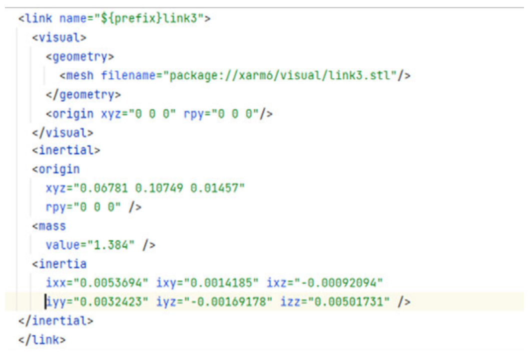
The result of image and point cloud acquisition, processing, and robot trajectory generation can be verified. The experiment parameters and other information can be visualized inside ROS simulation, which is helpful for algorithm modification and verification.

In order to visualize, simulate and control the motion of the xArm6 robot, it is necessary to build a robot-specific model in the ROS system. The format of the robot model in the ROS system is named URDF(Unified Robot Description Format), an XML-based robot description format that abstracts the robot model into a markup text file. This format can accurately describe the size, colour, mass and moment of inertia of the robot links, as well as the position and rotation angles of the robot joints.

Figure 2.20 (a) and (b) show the selected section of the URDF model of the xArm6 robot, and Fig. 2.20 (c) and (d) visualize the contents of the URDF model.

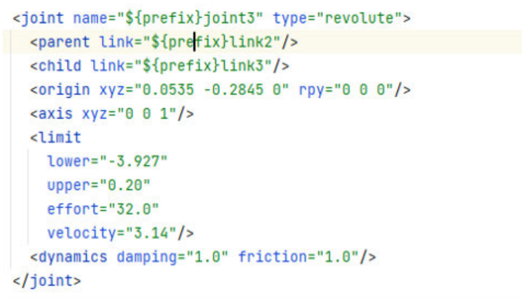
The <**link**> element describes the name, frame location, mass and inertia, visual features, and collision properties of the robot linkage. The <link> element contains multiple tags, whose functions are described below:

-<visual>: This tag specifies the shape of the link, which contains the file path of the 3D STL model of each link, when ROS loads the URDF file, the 3D model of



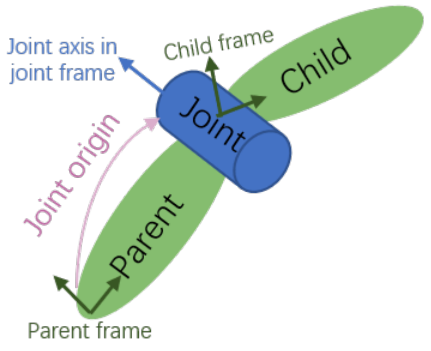
```
<link name="${prefix}link3">
  <visual>
    <geometry>
      <mesh filename="package://xarm6/visual/link3.stl"/>
    </geometry>
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </visual>
  <inertial>
  <origin
    xyz="0.06781 0.10749 0.01457"
    rpy="0 0 0" />
  <mass
    value="1.384" />
  <inertia
    ixx="0.0053694" ixy="0.0014185" ixz="-0.00092094"
    iyy="0.0032423" iyz="-0.00169178" izz="0.00501731" />
</inertial>
</link>
```

(a) URDF Link description

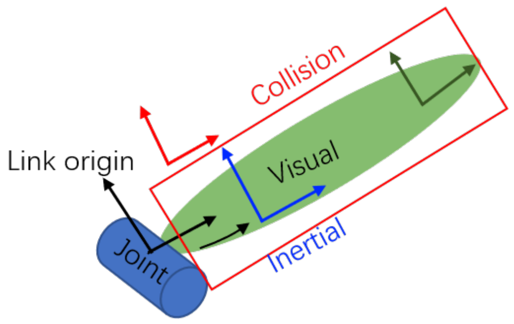


```
<joint name="${prefix}joint3" type="revolute">
  <parent link="${prefix}link2"/>
  <child link="${prefix}link3"/>
  <origin xyz="0.0535 -0.2845 0" rpy="0 0 0"/>
  <axis xyz="0 0 1"/>
  <limit
    lower="-3.927"
    upper="0.20"
    effort="32.0"
    velocity="3.14"/>
  <dynamics damping="1.0" friction="1.0"/>
</joint>
```

(b) URDF Joint description



(c) URDF Link image



(d) URDF Joint image

Figure 2.20: URDF Robot description format

the links will be automatically loaded and visualized.

-<collision>: The content of this tag is usually the same as the <visual> tag, but with another function for calculating the collision parameters of the robot. Its origin is the reference frame of the collision element.

-<inertial>: This tag is required when the robot model is loaded and simulated in a physic simulator (ROS Gazebo), the origin of this tag is the inertial reference frame, which is required to be at the center of the gravity of the link.

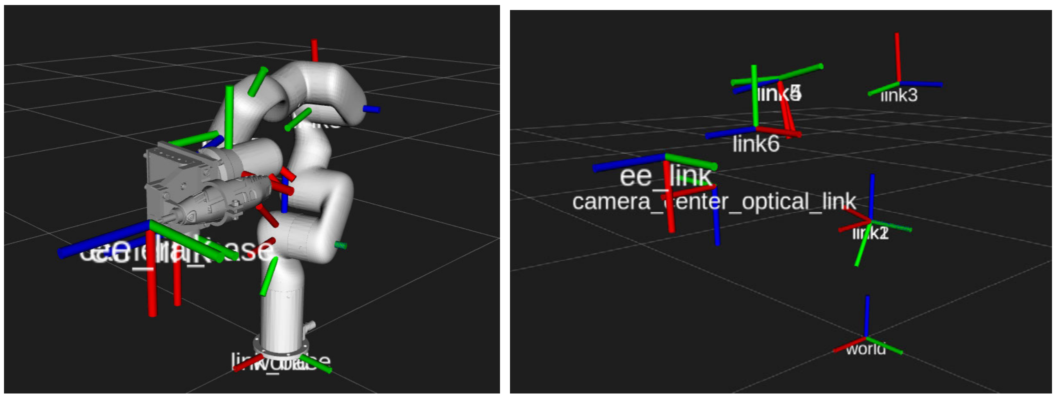
The <**joint**> element describes the relative relationship and connection method between two links, including the joint name, joint origin, joint limitation and other properties. The type of joint can be revolute, prismatic or planar. The <joint> element also contains multiple tags, whose functions are shown below:

-<origin>: The origin of the joint is located at the origin of the child link, which is the relative position of the child frame to the parent frame.

-<limit>: The joint's limit defines the joint's rotation and speed limitations.

From the above description, the URDF model mainly contains the position and rotation relationship between each link and joint of the robot. By using the D-H parameter of the xArm6 robot given in Table 2.2, the URDF model of the xArm6 robot can be established. Full text of the URDF model can be checked in the Appendix.

As shown in Fig. 2.21(a), the URDF model of the robot is visualized in the ROS Rviz system. The TF package [86] in the ROS system can also be used to verify the URDF model, as shown in Fig. 2.21(b), the TF package displays the tree structure of the robot in which the relationship between links and joints is visualized and can be checked easily. After comparing with the DH parameters, the URDF model of the robot is correctly established.



(a) xArm6 robot visulization in Rviz

(b) xArm6 robot TF figure

Figure 2.21: xArm6 robot URDF model

Beside the robot model, the OpenNI package [87] is uesd to create the simulated 3D camera in the ROS simulation environment with the same function as its real counterpart. This simulated camera is aligned onto the robot model by adding the camera description into the URDF model of the robot, including the resolution, installation position, reference coordinate frames, and other relevant parameters of the Microsoft Azure Kinect camera. Figure 2.22 shows the result of the simulated camera that can accurately collect the point cloud data of the object in the simulation environment.

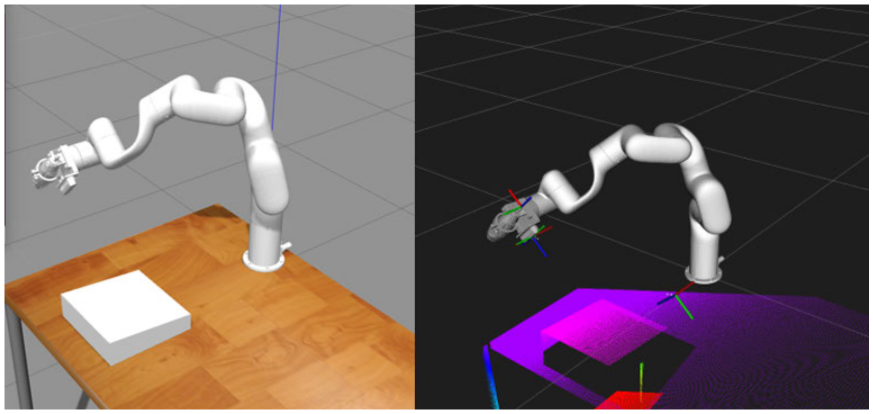


Figure 2.22: Simulated camera in ROS system

The final step is to load the crab model mentioned in Section 2.4.3 into the simulation environment, as shown in Fig. 2.23, the complete simulation environment has thus been built.

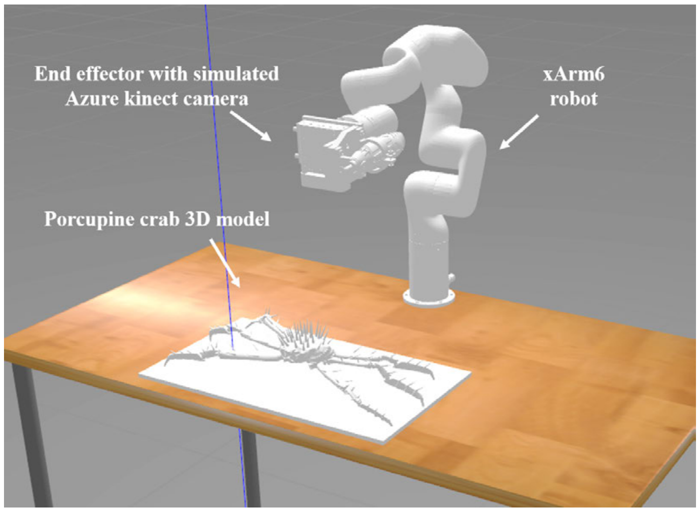


Figure 2.23: Complete simulation environment

2.7 Robot Control System Setup

Once the proposed method that is verified in the ROS simulation system, the ROS control system can send the control commands over the network to the robot control box to control the real robot arm in the laboratory. It can also collect the real-time status of the robot system while the real robot arm is executing the trajectory so that the operator can monitor all parameters of the experiment.

As shown in Fig. 2.24, the MoveIt [88] and the ros_control [89] packages are used to control the real robot. This thesis generates the robot spine removal path from the 2D image and 3D point cloud data, which will be imported into MoveIt package to calculate the joint parameters corresponding to all points on the robot tool path. The output joint parameters will be transmitted to the ros_control package to control the real robot to execute the trajectory.

The ros_control package contains a series of controller, actuator and hardware interfaces to control the real robot actuators. It uses a closed-loop feedback mechanism,

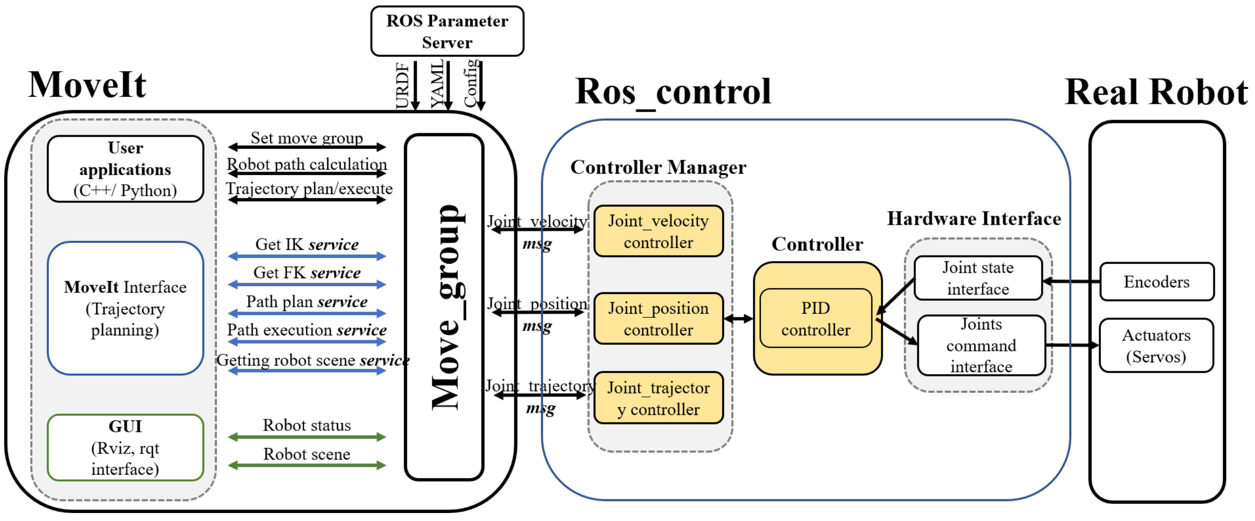


Figure 2.24: Robot control system diagram

typically the PID controller that sends the appropriate commands to robot actuators and receives real-time status from the robot joints' encoders for feedback control. The MoveIt and the ros_control packages will be described in detail in the following sections.

2.7.1 MoveIt Package

In this thesis, the robot's basic motion planning is implemented using the ROS MoveIt package, which has collision checking, motion planning, and inverse kinematics(IK) functions. The control structure diagram of the MoveIt package is shown in Fig. 2.25.

The move_group node is the core of the MoveIt package, as this node acts as an integrator of the various components of the ROS system and the robot controllers. From the structure diagram, it can be seen that the move_group node loads the robot kinematics data such as the robot description file (URDF), the semantic robot

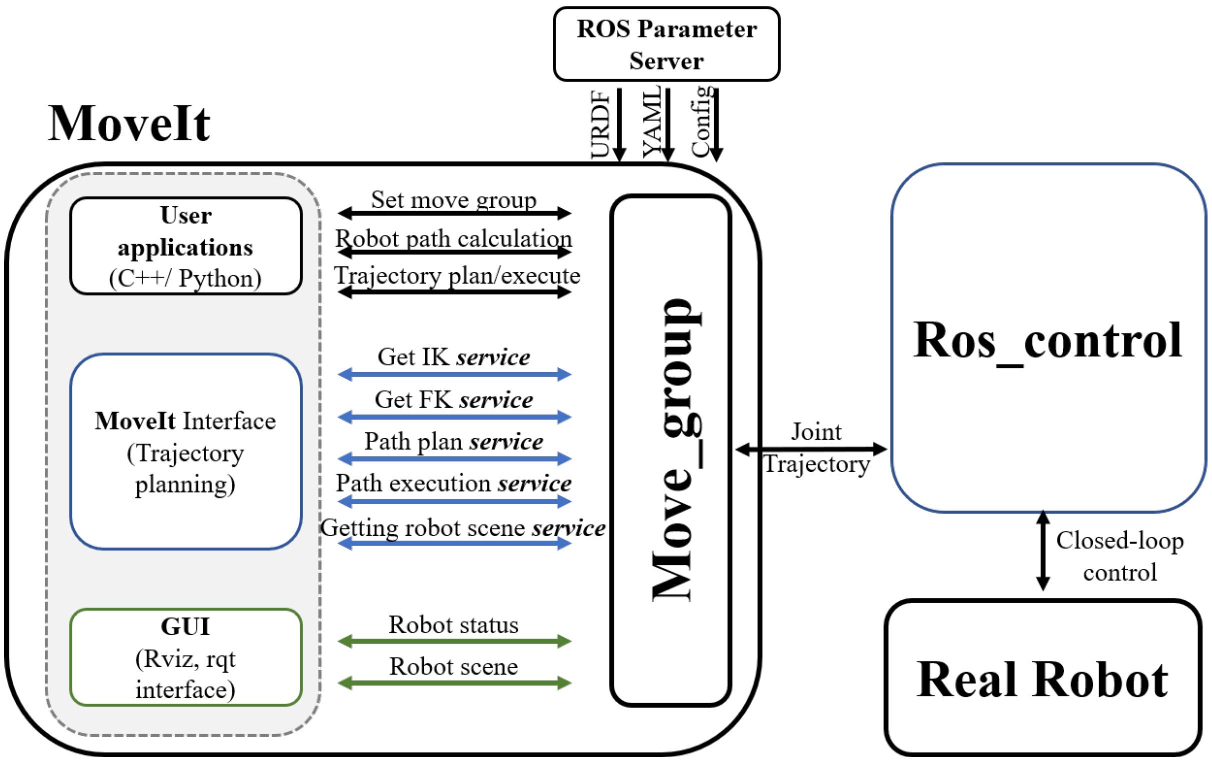


Figure 2.25: MoveIt package control structure diagram

description file (SRDF)[4] and other configuration files from the ROS parameter server, which will be used for motion planning and simulation by the MoveIt package. After the robot trajectory is generated, the trajectory will be sent from the move_group node to the ros_control package for execution. This node also collects the robot controller status, robot joints status and other datas from the real robot.

MoveIt package provides several sets of inverse kinematic(IK) solvers that can be used to solve the rotational angle of each joint of the robot when the end-effector pose is known. The default IK solver is the KDL IK solver distributed by the Orocos

[4]The SRDF complement the URDF and specifies joint groups, default robot configurations, additional collision checking information, and additional transforms that may be needed to completely specify the robot's pose.

Project [90], which is a numerical IK solver, MoveIt also provides the TRAC-IK optimized numerical solver [91] and the IK-FAST analytical solver [92], and can import the user-defined IK solver as well.

There are some stopping mechanism in the robot operational space to prevent the robot from colliding with itself and obstacles in the environment while moving. The MoveIt package provides the FCL(Flexible Collision Library)-based collision detection function [93], which is an open-source function that implements various collision detection and avoidance algorithms.

MoveIt package can communicate with multiple motion planners through a *plugin* interface using the ROS Action and Service communication protocol (offered by the move_group node). The primary function of the motion planner is to move the robot end-effector from one pose to a new pose with position, orientation, visibility, and joint constraints. The OMPL [94] is used as the default motion planner, which will be described in detail in the following sections.

Based on the above description, the MoveIt package has all the basic functions for robot motion planning. In this thesis, the calculated waypoints for spine removal are imported to the MoveIt package to plan the robot motion.

Figure 2.26 shows the general process of motion planning using the MoveIt package. Before using the MoveIt package for motion planning, the start pose and the end pose[5] of the robot end-effector need to be determined. Using the inverse kinematic (IK) solver, the rotational angle of each joint of the robot at the start and end poses can be calculated and imported into the motion planer. The goal of motion planning

[5]Pose: the pose contains the xyz coordinate of the tool center point (TCP point) in the global frame $\mathcal{B}$ and the rotational angles of the tool frame $\mathcal{T}$ with respect to the global frame $\mathcal{B}$

is to obtain a series of joint trajectories, velocities, accelerations, and other parameters from the start pose to the end pose while ensuring that the robot does not collide with itself or other obstacles. The result of motion planning will be sent back to the move_group node, which then is transferred to the ros_control package for execution.

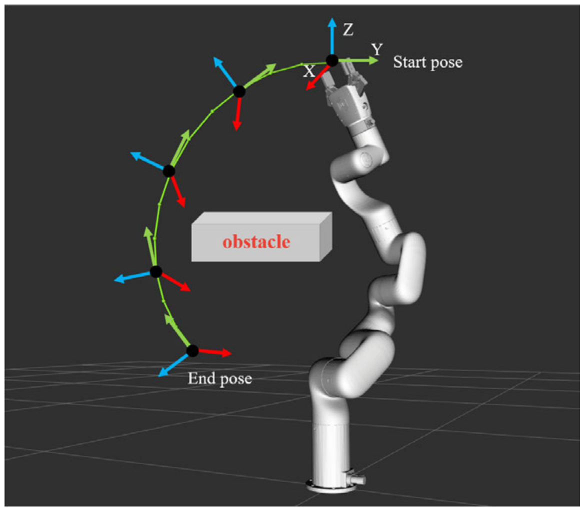
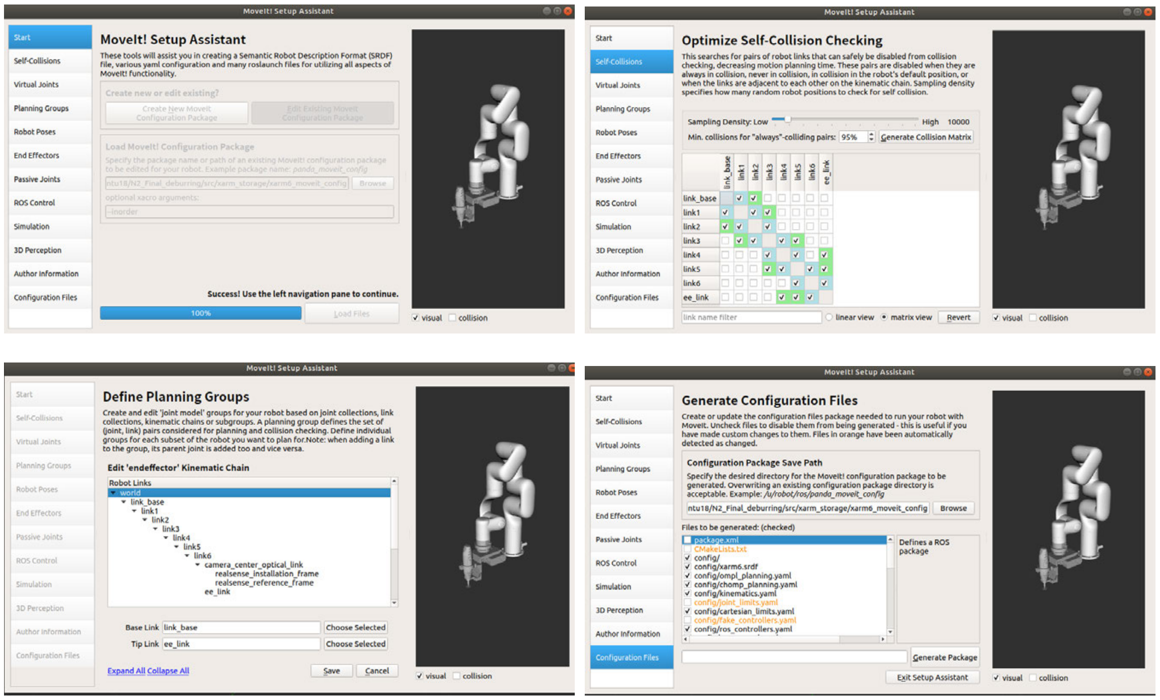


Figure 2.26: Motion planning using MoveIt package

2.7.1.1 MoveIt Setup Assistant Configuration

The previous section describes the main function of the MoveIt package. To use the MoveIt package control the robot, the robot model needs to be configured using the ROS-based MoveIt Setup Assistant plugin, a graphical user interface. The Unified Robotics Description Format(URDF) model of the robot is imported to this plugin. The Semantic Robot Description Format(SRDF) description file and other necessary configuration files will be generated, which will be used in the MoveIt package to simulate and visualize the robot's motion and can also be used to control the real robot arm.

The configuration process of the MoveIt setup assistant plugin is shown in Fig. 2.27, in which the left side of the interface shows the main contents that need to be configured. After the URDF model of the robot is loaded into the MoveIt setup assistant plugin, the next step is to configure the robot's self-collision detection matrix. The kinematic planning group also needs to be set up. In this thesis, the kinematic planning group is set as "xArm6", which contains six rotational joints. After that, the robot's start pose is set so that it can quickly return to the start pose at any time during its motion. After the above process is completed, a complete robot configuration package can be generated.



MoveIt! Setup Assistant

These tools will assist you in creating a Semantic Robot Description Format (SRDF) file, various yaml configuration and many roslaunch files for utilizing all aspects of MoveIt! functionality.

Create new or edit existing?

Create New MoveIt Configuration Package | Edit Existing MoveIt Configuration Package

Load MoveIt! Configuration Package

Specify the package name or path of an existing MoveIt! configuration package to be edited for your robot. Example package name: *panda_moveit_config*

ntu18/N2_Final_deburring/src/xarm_storage/xarm6_moveit_config | Browse

optional xacro arguments:

--inorder

Success! Use the left navigation pane to continue.

100% | Load Files

Optimize Self-Collision Checking

This searches for pairs of robot links that can safely be disabled from collision checking, decreasing motion planning time. These pairs are disabled when they are always in collision, never in collision, in collision in the robot's default position, or when the links are adjacent to each other on the kinematic chain. Sampling density specifies how many random robot positions to check for self collision.

Sampling Density: Low — High 10000

Min. collisions for "always"-colliding pairs: 95% | Generate Collision Matrix

	link_base	link1	link2	link3	link4	link5	link6	ee_link
link_base		✓	✓	☐	☐	☐	☐	☐
link1	✓		✓	✓	☐	☐	☐	☐
link2	✓	✓		✓	☐	☐	☐	☐
link3	☐	✓	✓		✓	✓	☐	☐
link4	☐	☐	☐	✓		✓	☐	✓
link5	☐	☐	☐	✓	✓		✓	✓
link6	☐	☐	☐	☐	☐	✓		✓
ee_link	☐	☐	☐	☐	✓	✓	✓	

link name filter | linear view [✓ matrix view] | Revert

Define Planning Groups

Create and edit 'joint model' groups for your robot based on joint collections, link collections, kinematic chains or subgroups. A planning group defines the set of (joint, link) pairs considered for planning and collision checking. Define individual groups for each subset of the robot you want to plan for.Note: when adding a link to the group, its parent joint is added too and vice versa.

Edit 'endeffector' Kinematic Chain

```
Robot Links
▾ world
  ▾ link_base
    ▾ link1
      ▾ link2
        ▾ link3
          ▾ link4
            ▾ link5
              ▾ link6
                ▾ camera_center_optical_link
                    realsense_installation_frame
                    realsense_reference_frame
                  ee_link
```

Base Link: link_base | Choose Selected

Tip Link: ee_link | Choose Selected

Expand All Collapse All | Save | Cancel

Generate Configuration Files

Create or update the configuration files package needed to run your robot with MoveIt. Uncheck files to disable them from being generated - this is useful if you have made custom changes to them. Files in orange have been automatically detected as changed.

Configuration Package Save Path

Specify the desired directory for the MoveIt! configuration package to be generated. Overwriting an existing configuration package directory is acceptable. Example: */u/robot/ros/panda_moveit_config*

ntu18/N2_Final_deburring/src/xarm_storage/xarm6_moveit_config | Browse

Files to be generated: (checked)

- ☐ package.xml
- ☐ CMakeLists.txt
- ✓ config/
- ✓ config/xarm6.srdf
- ✓ config/ompl_planning.yaml
- ✓ config/chomp_planning.yaml
- ✓ config/kinematics.yaml
- ☐ config/joint_limits.yaml
- ✓ config/cartesian_limits.yaml
- ☐ config/fake_controllers.yaml
- ✓ config/ros_controllers.yaml

Defines a ROS package

Generate Package

Exit Setup Assistant

Figure 2.27: MoveIt setup assistant configuration process

In order to verify the accuracy of the generated configuration package, the ROS RViz visualization tool is used to load the configuration packages generated by the MoveIt setup assistant plugin. As shown in Fig. 2.28, the 3D model of the xArm6 robot and the frame location for every link are correctly displayed in the RViz tool, which indicates that the configuration package is ready to use.

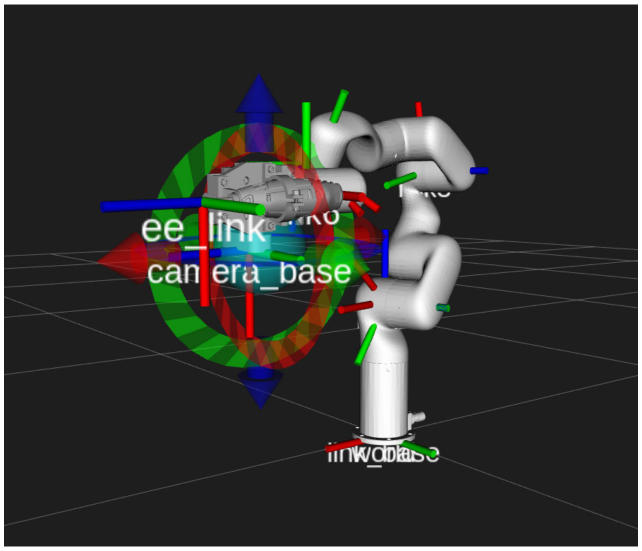


Figure 2.28: MoveIt setup assistant configuration result

2.7.1.2 Inverse Kinematic(IK) Solvers

The inverse kinematic (IK) is the mathematical process of calculating the joint parameters needed to let the robot end-effector be in a given orientation and position related to the global coordinate system.

The Kinematics and Dynamics Library(KDL) solver [90] is the default IK solver in the MoveIt package, which is encapsulated in the *MoveIt_kdl_parser* package that provides both the forward kinematic (FK) and the IK functions for robot motion planning. The KDL IK solver is developed based on the Newton-Raphson iteration method [95], which has the advantage of fast convergence speed and self-correcting capability.

The main idea of the Newton-Raphson method is to establish a relationship between the robot velocities in joint space with the robot end-effector velocities in the task space. As shown in eq. (2.11), the generalized velocities of the robot end-effector in task space consists of the velocity $\mathbf{v}$ and angular velocity $\mathbf{w}$, a six-dimensional vector. It can be calculated by using eq. (2.11), where $\mathbf{D} = [\vec{\mathbf{d}}^{\tau}, \delta^{\tau}]$ consists of instantaneous linear velocity $\vec{\mathbf{d}}$ and angular velocity δ, which can be calculated from the derivative of the generalized velocity with respect to time t.

$$\begin{pmatrix} \mathbf{v} \\ \mathbf{w} \end{pmatrix} = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \begin{pmatrix} \mathbf{d} \\ \sigma \end{pmatrix} = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \mathbf{D} \tag{2.11}$$

The generalized velocity of the six-degree-of-freedom robot in the task space and the velocity in joint space can be inter-converted by a 6×6 Jacobian matrix $\mathbf{J}$, substituting $\mathbf{J}$ into eq. (2.11) leads to eq. (2.12).

$$\begin{pmatrix} \mathbf{v} \\ \mathbf{w} \end{pmatrix} = \mathbf{J}(\boldsymbol{\theta}) \frac{d\theta}{dt} = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \mathbf{D} \tag{2.12}$$

Equation(2.12) can be re-written as $\mathbf{J} d\theta = D$, where $\mathbf{J}$ is the Jacobian matrix of the xArm6 robot which is a function of joint angles $\theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$. If the Jacobian matrix $\mathbf{J}$ is of full rank, the equation will have infinite sets of solutions. However, there will be no solution when the Jacobian matrix $\mathbf{J}$ is singular [95]. Therefore, to avoid that, the Moore-–Penrose pseudoinverse [96] of the Jacobian matrix is calculated. Using the Householder Transformation and the singular value decomposition(SVD) method to decompose **J** (eq. (2.13)). The pseudoinverse of **J** is shown in eq. (2.14). Equation (2.12) can be re-written as eq. (2.15).

$$\mathbf{J} = \mathbf{U}\sum V^* \tag{2.13}$$

$$\mathbf{J}^+ = V\sum^{+}\mathbf{U}^* \tag{2.14}$$

$$d\boldsymbol{\theta} = \mathbf{J}^+\mathbf{D} \tag{2.15}$$

Assume that the current robot pose is $\mathbf{T}_{\text{cur}}$, Δ can be obtained from the instantaneous velocity matrix of the robot, which is shown in eq. (2.16) and eq. (2.17), where I is a 4-by-4 identity matrix. **D** in eq. (2.15) can be obtained by Δ post-multiply $\mathbf{T}_{\text{cur}}$.

$$\Delta = \text{Trans}(dx, dy, dz) \times \text{Rot}(k, d\theta) - I \tag{2.16}$$

$$\Delta = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -\delta x & \delta y & 0 \\ \delta z & 1 & \delta z & 0 \\ -\delta y & \delta x & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -\delta x & \delta y & dx \\ \delta z & 0 & \delta z & dy \\ -\delta y & \delta x & 0 & dz \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.17}$$

Set $\mathbf{T}_{\text{end}}$ as the target pose, and $\mathbf{T}_{\text{cur}}$ as the current pose of the robot, the correlation between the $\mathbf{T}_{\text{cur}}$ and $\mathbf{T}_{\text{end}}$ is shown in eq. (2.18).

$$\mathbf{T}_{\text{end}} = \mathbf{T}_{\text{cur}}(\Delta + I) \tag{2.18}$$

$$\Delta = \mathbf{T}_{\text{cur}}^{-1}\mathbf{T}_{\text{end}} - I \tag{2.19}$$

Combining (2.17) and (2.19), the value of $\mathbf{D} = [\vec{d^{\tau}}, \delta^{\tau}]$ can be calculated. In summary, the main steps for solving the IK of a general 6-DOF robot using the Newton-Raphson method are described below:

(1) Based on the current joint angles, calculate the current pose of the robot end-effector $\mathbf{T}_{\text{cur}}$ using the FK.

(2) Set $\mathbf{T}_{\text{end}}$ as the target pose of the robot end-effector, and calculate the instantaneous velocity matrix by using eq. (2.19). Then calculate the instantaneous velocity vector $\mathbf{D}$ of the robot.

(3) Calculate the Jacobian matrix $\mathbf{J}$ of the robot by taking the derivative of each joint angle, and then establish the equation of the Newton-Raphson method $\mathbf{J}d\theta = \mathbf{D}$.

(4) Use the SVD decomposition method to decomposite the Jacobian matrix $\mathbf{J}$ and get the pseudoinverse $\mathbf{J}^{+}$, Calculate the value of $d\theta$ with the equation $d\theta = \mathbf{J}^{+}\mathbf{D}$.

(5) Calculate the value of $||d\theta||$, and set a value of motion planning accuracy ϵ. If $||d\theta|| \leq eps$, exit the loop and output the current pose of the end-effector as $\mathbf{T}_{\text{cur}}$. If the iteration count is larger than the limitation count N, exit the loop and output the failure message. Otherwise, set $\theta_{cur} = \theta_{cur} + d\theta$, and calculate the value of $\mathbf{T}_{\text{cur}}$, repeat steps 1 to 5, until the loop exit condition is met. θ will be the final solution of the inverse kinematic(IK) calculation.

TRAC-IK [97] is similar to the KDL inverse kinematic solver, which is also a numerical solver with many improvements. It concurrently calculates the robot IK with sequential quadratic programming(SQP) IK algorithm and the improved Newton-Raphson iteration IK algorithm, with the optimum trajectory result returned. The TRAC-IK solver is about twice as fast as the KDL solver, and the success rate increases from 70% to 98% [97]. The xArm6 robot is a six degree-of-freedom robot arm, which requires high stability for the inverse kinematic calculation, so the TRAC-IK solver is chosen in this thesis.

2.7.1.3 Collision Detection

Collision detection is also an important part of robot trajectory planning with the MoveIt package, a criterion for verifying the generated robot trajectory. Traditional collision detection algorithms are computationally complex and time consuming [98]. The MoveIt package has a built-in collision detection library based on the Flexible Collision Library(FCL) [93], which can detect collisions between basic shapes (such as sphere, cubes, cylinders, etc.) and has a high efficiency. It is ideal for collision detection during robot trajectory planning. When using the FCL method for collision detection, multiple sampling points need to be checked during the trajectory planning process to determine whether the joint angle exceeds the limit and whether the robot has collision with itself or with other obstacles in the environment.

In the MoveIt package, the Aixe Align Bounding Box method (AABB method) is adopted, which uses a rectangular parallelepiped with each face perpendicular to one of the basis vectors in the global coordinate frame. The fumula of the AABB method is shown in eq. (2.20), where the $l_{\min}, m_{\min}, n_{\min}$ are the minimum coordinate and $l_{\max}, m_{\max}, n_{\max}$ are the maximum coordinate of the AABB box.

$$X = \{(x, y, z) \mid l_{\min} < x < l_{\max}, m_{\min} < y < m_{\max}, n_{\min} < z < n_{\max}\} \tag{2.20}$$

Then the Separating Axis Theorem [99] is used to determine whether two objects intersect. The process is as follow: the ranges of the projection of the object A and B along X-axis in the Cartesian coordinate system are set as (min_{A_x}, max_{A_x}) and (min_{B_x}, max_{B_x}). If these two ranges intersect, object A and object B may intersect along the X-axis. The same process can be performed along Y-axis and Z-axis. Only when the intersection occurs in all these three directions, it can be confirmed that object A and object B collide with each other.

When using the MoveIt package to detect the collision of the robot, the detection process mainly inculdes the self-collision detection and the collision detection with the external obstacles in the nearby environment.

A. Self collision detection

The robot self-collision detection matrix has been configured in Section 2.7.1.1, the effect of self-collision detection is shown in Fig. 2.29, where the part marked in red indicates the location of the collision. The pseudocode of the self-collision detection is shown in Algorithm 1.

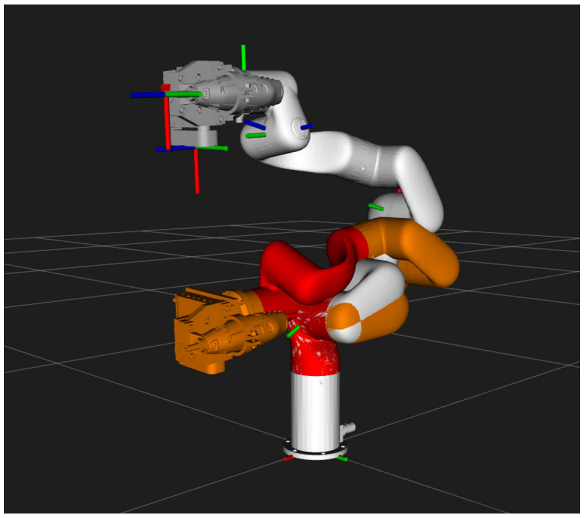


Figure 2.29: Robot self-collision detection

B. Collision with external obstacles detection

As shown in Fig. 2.30, a rectangular box object is added next to the robot to verify the effectiveness of the collision detection between the robot and the external obstacles, a simulation environment is created using the ROS RViz package. Then, while manually dragging the robot arm in the simulation environment for the collision detection test, the interfering parts will turn red when the robot arm interferes with the box object, showing that the collision is detected.

As shown in Fig. 2.31, to test the collision detection and avoidance trajectory planning function in the MoveIt package, two poses of the robot arm are set in the ROS Rviz simulation environment as *Pose1* and *Pose2*.

We set *Pose1* as the robot start pose and *Pose2* as the robot target pose, and use the TRAC-IK inverse kinematic solver in the MoveIt package for trajectory planning. When there is no obstacle in the robot trajectory, the planned trajectory of the robot is shown in Fig. 2.32(a). Then, a cubic object is added to the simulation environment

Algorithm 1: Robot self collision detection

```
1 Forward_Kinematic($x$);
2 for $k \leftarrow 1$ to 6 do
3     $C_k \leftarrow$ FCL_CUBE_CREATE($x$,$k$);
4     if _CUBE_CREATE($x$,$k$) then
5         Return Trapped
6     end
7     Return Advanced
8 end
```

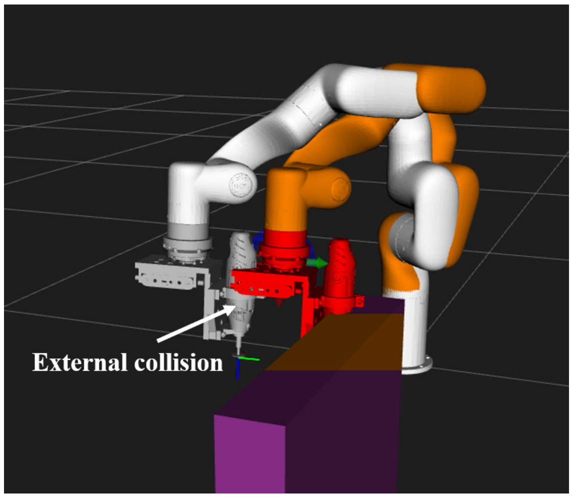


Figure 2.30: Robot external collision detection

as an obstacle to the robot's motion. The robot trajectory is planned from *Pose1* to *Pose2*. The updated robot trajectory is shown in Fig. 2.32(b), it can be seen that the robot trajectory successfully avoids the obstacle. The effectiveness of the collision detection and obstacle avoidance algorithm can be verified.

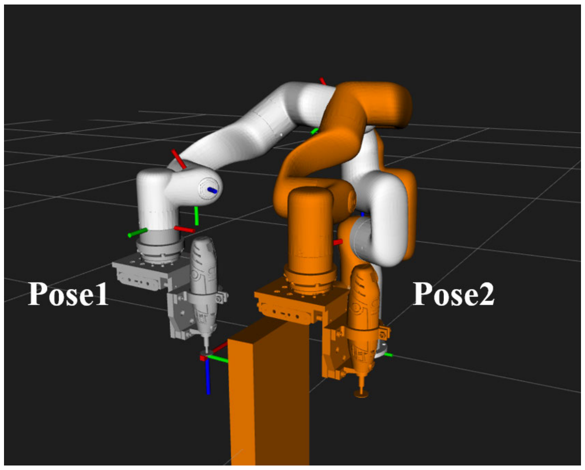
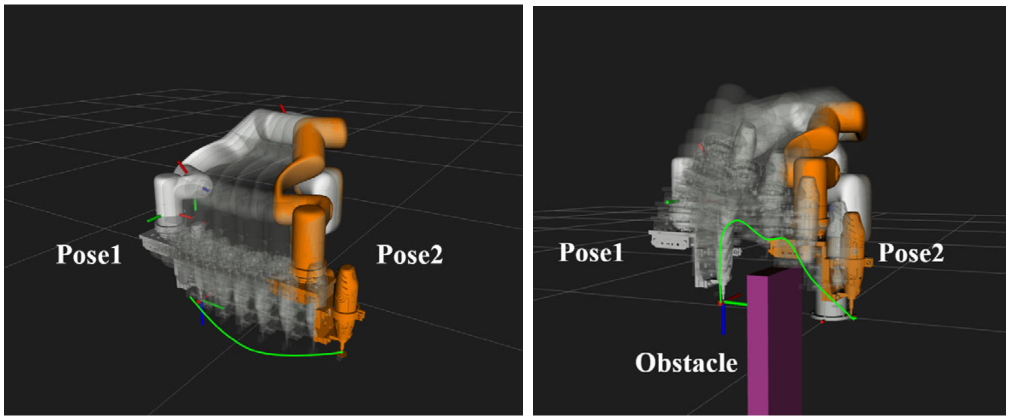


Figure 2.31: Two poses of the robot in RViz simulation environment



(a) Without obstacle (b) With obstacle

Figure 2.32: Trajectory planning result with and without obstacle

2.7.2 ROS Control Package

The ros_control package is the connection layer between the ROS system in the central control PC and the real robot controller. It contains a series of controller interfaces, actuator interfaces, hardware interfaces and multiple practical toolboxes and can help researchers quickly develop their projects and improve development efficiency, simplify the coding and commissioning workload. The ros_control package contains many modules, such as the classical PID control modules. In this thesis, ros_control is used to connect the ROS system to the xArm6 robot controller and control the xArm6 robot arm's trajectory. The control system diagram of the ros_control package to the real robot controller is shown in Fig. 2.33.

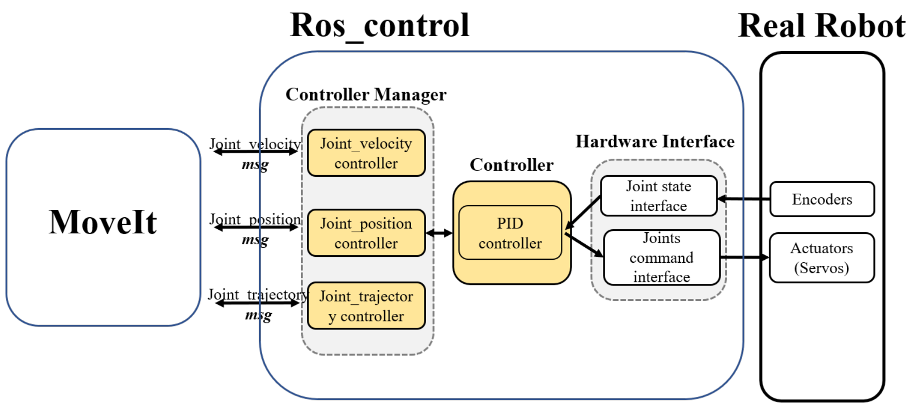


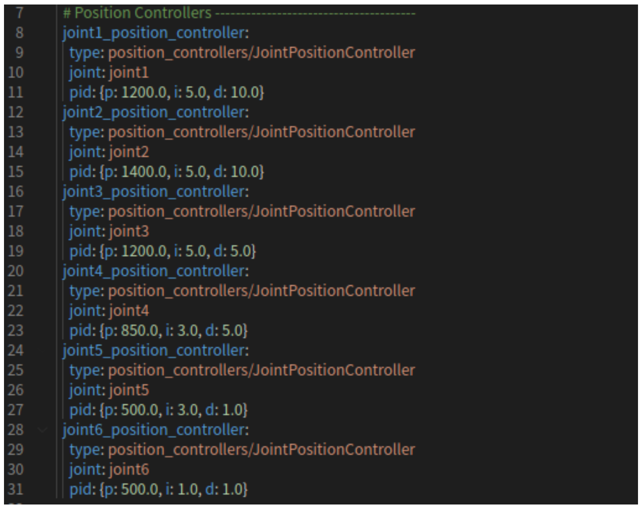
Figure 2.33: ROS_control diagram in this thesis

2.7.2.1 ROS Control Controller

The controller in the ros_control package receives the expected trajetory of each joint of the robot and uses this trajectory info to control the real robot arm. The joint trajectory data can be either manually inputted or generated from the function packages in ROS, such as the MoveIt package. In this thesis, the joint_trajectory_controller is used as the main controller of the robot.

The joint_trajectory_controller is a controller in the ros_control package for executing a trajectory of a set of joint values in the robot joint space, where the trajectory is a set of path points consisting of position, velocity, acceleration and time stamp values. By default, the controller internally provides a B_spline [100] interpolation algorithm and can also be integrated with various kinematic planners.

For the xArm6 robot arm to be controlled by the ros_control package, the control description file needs to be configured in the ROS environment, including the robot type, joint parameters, network communication rate and other parameters of the xArm6 robot. The ros_control package will load this configuration file, allocate the necessary system resources, check for conflicts, and configure the associated resource related to robot control. The control description file of the xArm6 robot is shown in Fig. 2.34.



```
# Position Controllers ---------------------------------------
joint1_position_controller:
  type: position_controllers/JointPositionController
  joint: joint1
  pid: {p: 1200.0, i: 5.0, d: 10.0}
joint2_position_controller:
  type: position_controllers/JointPositionController
  joint: joint2
  pid: {p: 1400.0, i: 5.0, d: 10.0}
joint3_position_controller:
  type: position_controllers/JointPositionController
  joint: joint3
  pid: {p: 1200.0, i: 5.0, d: 5.0}
joint4_position_controller:
  type: position_controllers/JointPositionController
  joint: joint4
  pid: {p: 850.0, i: 3.0, d: 5.0}
joint5_position_controller:
  type: position_controllers/JointPositionController
  joint: joint5
  pid: {p: 500.0, i: 3.0, d: 1.0}
joint6_position_controller:
  type: position_controllers/JointPositionController
  joint: joint6
  pid: {p: 500.0, i: 1.0, d: 1.0}
```

Figure 2.34: xArm6 robot ros_control configuration file

2.7.2.2 Data Transmission Setting

ROS generally provides bi-directional data transmission through the Publisher/Subscriber and Service/Client protocols [101]. However, when a stable and high-speed connection needs to be established between the ROS system and the real robot controller, the first two data transmission protocols cannot meet the requirements [101]. Therefore, this thesis uses the more advanced Action/Client data transfer protocol, which creates a system-level communication service with high privileges and can occupy the system resource to ensure stable data transfer between the ROS system with the real robot controller. The diagram of the ROS Action/client protocol is shown in Fig. 2.35.

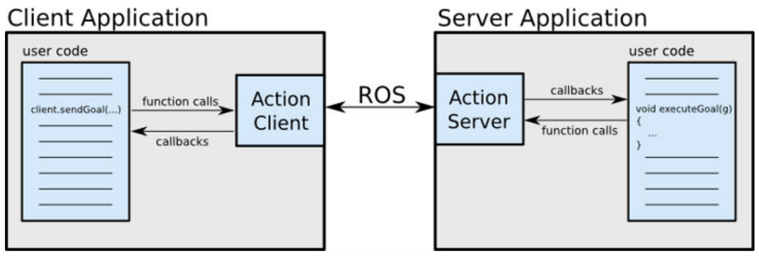


Figure 2.35: ROS Action/Client Protocol

2.8 Summary

This chapter described the overall design of the robot-based porcupine crab spine removal system. Firstly, the system design philosophy and requirements are introduced and analyzed, based on which the selection of each major part is determined. Next, the mechanical design of the robot working platform, the crab model and the robot end-effector are introduced in detail, and the system's overall electrical design is also elaborated. After that, the simulation platform is built in the ROS simulation environment, which is consistent with its real counterpart. Finally, the algorithm of robot trajectory planning with the IK and collision recognition are introduced. The real-time control requirement of the system is analyzed, and the robot control framework based on the ros_control package is built.

Chapter 3

Methodology

3.1 Introduction

[1]In this chapter, the core methodologies developed within our research are introduced in details, which are organized as follows. In Section 3.2, the point cloud pre-processing method is introduced, including the filtering and denoising, smoothing and normal finding, point cloud registration and reconstruction methods. Section 3.3 provides two trajectory planning methods for the porcupine spine removal application. The first method uses the point cloud data of the crab as the only data source and generates a robot spine removal trajectory covering the entire outer surface of the crab shell through a novel point cloud slicing method. This method has been vali-

[1]The method presented in this section is published in the conference paper [74], Haodong Wu and Dr.Ting Zou devised the project, worked out almost all of the technical details, and performed the simulation and experiment, and also wrote the paper with input from all authors. Heather Burke and Stephen King provided the background information and assisted with the on-site measurement of the research object. Brian Burke provided advice from the perspective of a fisheries expert.

dated in a simulation environment, but the cutting efficiency is relatively low because the trajectory contains many non-cutting waypoints. One the other hand, the second method combines the 2D image and 3D point cloud data to generate the trajectory. Aruco code, acting as a medium, is used to establish the relationship between a 2D image and a 3D point cloud data, so that the 3D coordinate of all the spines is able to be extracted. Moreover, the trajectory can be generated and optimized by adopting the cubic curve interpolation method.

3.2 Point Cloud Data Pre-processing

As introduced in Chapter 1, the Microsoft Azure Kinect RGB-D camera is used in this thesis as the point cloud data acquisition device. This camera is based on time-of-flight technology to measure the depth information of objects relative to the camera coordinate system. The camera emits laser with a specific frequency to the target area, and the object's depth information can be obtained by calculating the time from the emission to the return of the laser. The built-in algorithm can convert the depth information into the point cloud data used in this thesis. Due to the environment and other internal and external factors, such as the ambient light and vibrations, the raw point cloud data may include outliers, noises and other unavoidable errors. Therefore, the point cloud data pre-processing is significant to filter out the irrelevant information and ensure that the point cloud data only include the relevant points and information of the crab object.

3.2.1 Filtering and Denoising

Due to the wide field of view of the Microsoft Azure Kinect camera, the redundant information contained in raw point cloud data contains should be removed. A three-dimensional low pass filter is applied, which can remove point cloud data outside a given boundary along x, y, and z coordinates. In this thesis, the boundary is chosen as the edge of the platform where the crab model is placed, all point cloud data outside the platform is removed. Afterward, a statistical outlier removal algorithm based on Gaussian distribution($d \sim N\left(\mu, \sigma^2\right)$) is used to search and remove outliers as eq. (3.1).

$$\mu = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} d_{nm}, \quad \sigma = \sqrt{\frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m}\left(d_{nm}-\mu\right)^2} \tag{3.1}$$

where m and n are the column and row sequential numbers of each point, μ and σ are the mean and the standard deviation of the average distance from the point to its neighboring point inside the selected area [102]. As per the three-sigma rule of the Gaussian distribution[2], if the average distance from a point to its neighbouring points is outside the range between $\mu - 3\sigma$ and $\mu + 3\sigma$, this point will be considered as an outlier and needs to be removed. This approach can not only filter out the noises and outliers, but also have promising effects in removing the spines of the porcupine crabs in our study. By taking the distance of each point on the shell to its neighbouring points, those points on the spines that are far from the points on the shell are filtered

[2]The three-sigma rule denotes that, in probability theory and mathematical statistics, an event is considered to be practically impossible if it lies in the region of values of the normal distribution of a random variable at a distance from its mathematical expectation of more than three times the standard deviation.

out, thereby only point cloud data of points on the crab shell are retained. Figure 3.1 shows the point cloud comparisons before and after filtering and denoising.

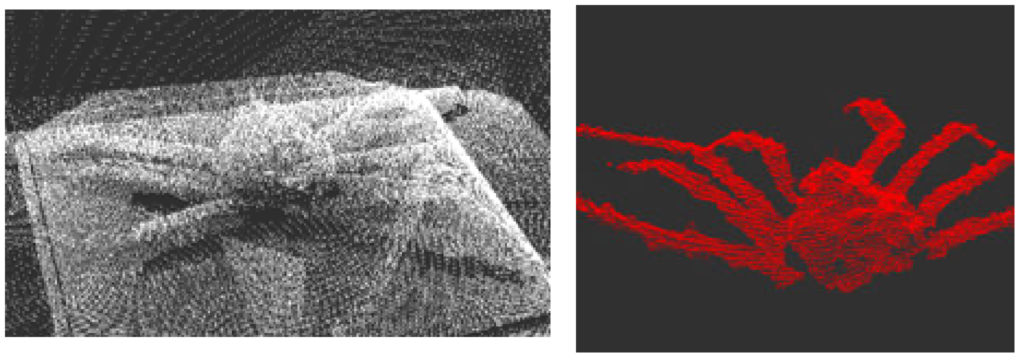
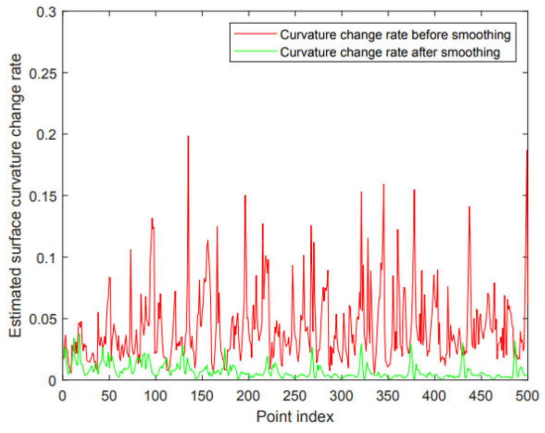

(a) Raw point cloud data (b) Point cloud data after low pass filter

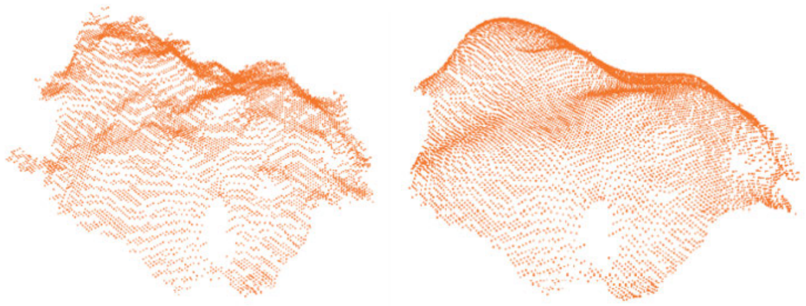
Figure 3.1: A comparison on the point cloud data before and after filtering and denoising

3.2.2 Smoothing and Calculating Surface Normal

The measurement and alignment errors of the camera, though minor, can lead to noises in the point cloud data that are difficult to be removed by using the statistical filtering and denoising method. Also, high-frequency point data may cause the change rate of the surface curvature significantly high. Moreover, since the spine data is removed in the filtering and denoising, holes are left on the surface of the crab shell. A least squares regression method [103] is adopted to smooth the point cloud data and fill the holes on the shell surface. Figure 3.2(a) illustrates the comparison of the curvature change rate of each tiny plane being composed of a center point and its neighboring points in the point cloud data before and after smoothing. It reveals that the curvature change rate significantly decreases after smoothing. The curvature rate is calculated by means of solving the eigenvalues of eq. (3.2)[104]. Figure 3.2(b) and (c) show the point cloud data of the crab base shell before and after smoothing, respectively. It is apparent that, after smoothing, the holes on the shell are smoothly filled.



(a)



(b) (c)

Figure 3.2: (a) The rate of change in curvature of the crab shell base before and after smoothing; (b) and (c): the point cloud data of the crab shell before and after smoothing.

During the spine removal process, the robot tool direction should be collinear to the normal vector of the surface point, aiming to improve the deburring efficiency without causing any damage to other parts of the crab shell. In this thesis, the principal component analysis (PCA)-based method [105] is used to calculate the normal vector of each surface point in the point cloud data. This method generates a tiny plane containing each point and its neighbouring points, the normal vector of the surface point thus being approximated as that of the tiny plane. The normal vector of the central point P_i can be estimated from solving the eigenvalues and the corresponding eigenvectors from the 3×3 neighborhood covariance matrix $\mathbf{C}$, defined as

$$\mathbf{C} = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbf{p}_i - \overline{\mathbf{p}}\right) \left(\mathbf{p}_i - \overline{\mathbf{p}}\right)^T \tag{3.2}$$

where $\mathbf{p}_i$ is the position vector of point P_i, N being the number of points neighbouring to P_i, $\overline{\mathbf{p}}$ being the position vector of the centroid point. The corresponding eigenvector of the minimum eigenvalue of matrix $\mathbf{C}$, defined as $\mathbf{n}_i = [n_{x_i}, n_{y_i}, n_{z_i}]^T$, is taken as the normal vector of P_i. Hence, following this approach, the normal vector of each point on the crab shell can be obtained, as shown in Fig. 3.3. Those normal vectors, along with the shell point coordinate, are used to calculate the robot tool path.

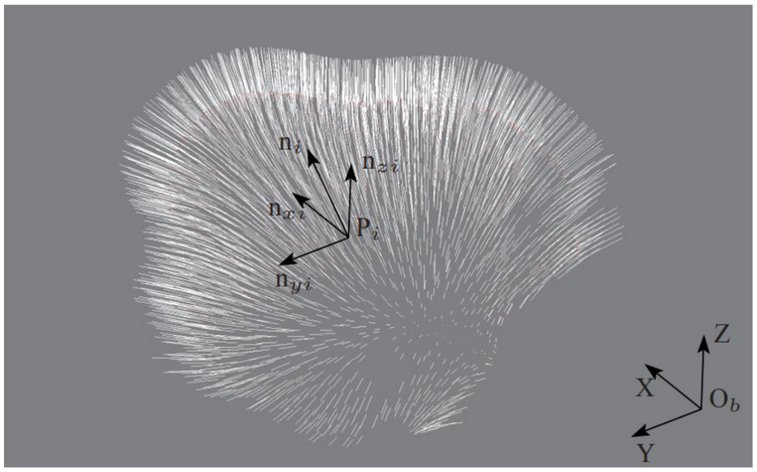


Figure 3.3: Normal vectors of surface points on the porcupine crab shell

3.2.3 Point Cloud Registration and 3D Reconstruction

Due to the principle of point cloud acquisition, one captured point cloud file only contains a set of points of the crab object surface in the line of sight of the camera. By using the KinectFusion [106] method, this thesis acquires multiple sets of point cloud files around the crab body for registration to find the relationship among each point cloud file, and then reconstructs a whole point cloud file containing all the information around the crab shell.

The 3D reconstruction method consists of several steps: the instantaneous camera poses of the point cloud files processed in former sections can be calculated by using the Iterative closest point(ICP) [107] algorithm. The truncated signed distance function(TSDF) that representing each point cloud file is then calculated. After that, using the camera pose information with the TSDF representation, the point cloud files under camera coordinate system $\mathcal{C}$ are converted to the global coordinate system $\mathcal{B}$ for registration and 3D reconstruction. The overall working flow of the KinectFusion method is shown in Fig. 3.4.

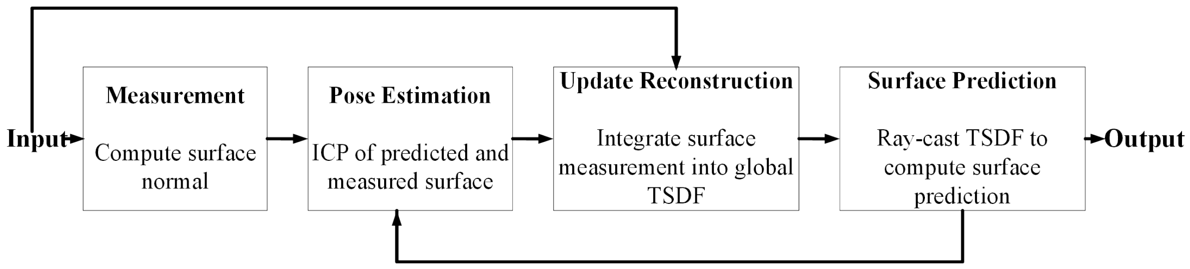

Input
Measurement
Compute surface normal
Pose Estimation
ICP of predicted and measured surface
Update Reconstruction
Integrate surface measurement into global TSDF
Surface Prediction
Ray-cast TSDF to compute surface prediction
Output

Figure 3.4: KinectFusion method working flow

3.2.3.1 SDF and TSDF

The Signed Distance Function (SDF) [108] represents the weighted signed distance value of each point in the grid along the direction of the camera's line of sight to the object surface. The absolute value of the SDF for each point indicates the perpendicular distance between this point to the object surface. The points within the surface can be set as negative SDF values. Accordingly, positive SDF values will be given for the points allocated outside the surface. To extract the surface of the object, all points with zero SDF value should be identified and connected. As shown in Fig. 3.5, the curve represents all the points with zero SDF value, which are called Zero-crossing points.

For each point, its three-dimensional coordinates (X_w, Y_w, Z_w) in the global frame are first obtained, and the point projection coordinates (u, v) onto the image plane are calculated according to the pinhole camera model theory [109]. As shown in eq. (3.3), the difference between the calculated depth value of the projected point and the depth information captured by the camera is defined as the directed distance d, where z

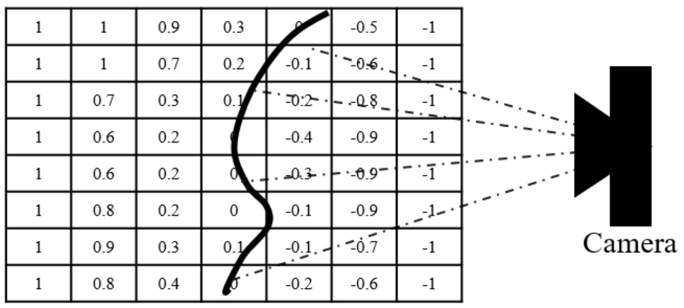
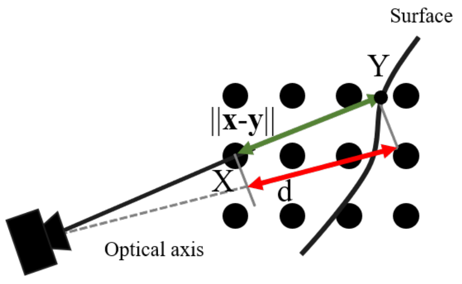


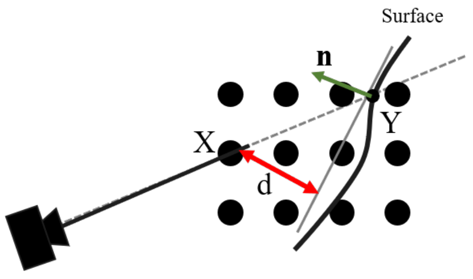
Figure 3.5: SDF surface representation

is the z-axis component of the three-dimensional coordinates of the point, and I_d is the depth image (point cloud) acquired by the camera. As shown in Fig. 3.6(a), the length d of the red line represents the SDF value of the point X. If d is negative, the point is in front of the surface; otherwise, it is behind the surface.

$$d = z - I_d(u, v) \tag{3.3}$$



(a) Point-to-Point distance



(b) Point-to-Plane distance

Figure 3.6: Illustration of the SDF calculation [106]

However, the high change rate of the surface curvature may adversely affect the accuracy of the SDF calculation by point-to-point distance method [108]. The tangent distance from the point to the object surface should be chosen for SDF calculation as this method will create a more accurate and robust result. As shown in Fig. 3.6(b), the vector $\mathbf{n}$ (green line) represents the surface normal vector, and d (red line) is defined as the distance from the point X to the tangent surface. Let (u, v) be the coordinates of point X in the camera coordinate system, and $\mathbf{n}(u, v)$ as the surface normal vector. The distance from the point to the tangent plane is selected as the optimized SDF value and calculated by eq. (3.4), in which $\boldsymbol{x}$ is the vector from the camera to point X and $\boldsymbol{y}$ is the vector from the camera to the surface point Y along the camera optical axis.

$$d = (\boldsymbol{y} - \boldsymbol{x})^T \boldsymbol{n}(u, v) \tag{3.4}$$

The method described in the previous section is a simple and efficient way for SDF calculation. Nevertheless, if the point is far away from the object surface, the accuracy of SDF value is not ideal. Fortunately, during the 3D reconstruction of the object, only the points near the object's surface will be factored. In contrast, the points far away from the object's surface are considered as outliers and will impose no effect on the 3D reconstruction process. Therefore, to avoid significant errors and reduce the computational load, the SDF values of these points need to be truncated. The SDF values within $|d| \leq \delta$ are named as the Truncated Signed Distance Function representation (TSDF) [110], as shown in eq. (3.5).

$$d_{\text{trunc}} = \begin{cases} -\delta, \text{ if } d < -\delta \\ d, \text{ if } |d| \leq \delta \\ \delta, \text{ if } d > \delta \end{cases} \tag{3.5}$$

3.2.3.2 Camera Pose Estimation and Point Cloud Reconstruction

As shown in Fig. 3.7. The ICP [107] is a classical point cloud registration algorithm. It calculates the transformation matrix to map one set of point cloud data to another one. The corresponding points in these two sets of point cloud data can match onto the other. The sampling frequency of the Azure Kinect camera is 60 Hz, and there is only a small movement between the continuously collected point cloud data, ensuring the ICP algorithm's convergence and accuracy.

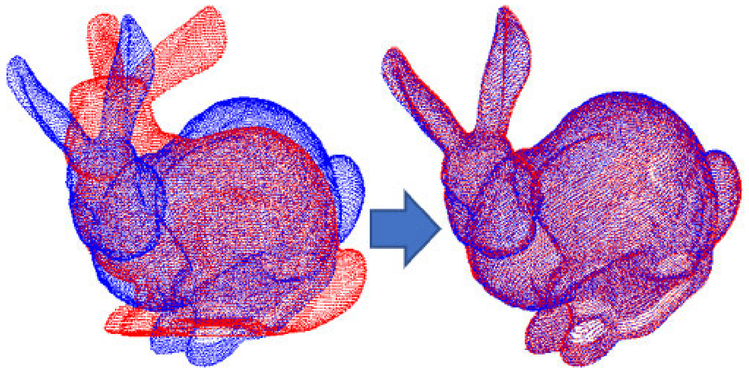


Figure 3.7: Point cloud registration using ICP algorithm [107]

The core algorithm of the ICP method is to use the energy minimization [107] to find the transform matrix between the camera coordinate system and the world/global coordinate system at the K-th frame. The ICP method is encapsulated in the open sourced point cloud library (PCL) [111], which can be easily integrated into the whole project.

After the camera poses are calculated for each set of point cloud data, the newly acquired point cloud data need to be transformed into the global coordinate frame and fused into the existing point cloud model. For example, let the coordinate of a point in the point cloud data that acquired in the $k-1$ frame be $F_{k-1}(\mathbf{p})$, and the corresponding point in the point cloud data in frame k be $F_{R_k}(\mathbf{p})$. If these two corresponding points are located in the same voxel, the coordinates of these two points are fused by using the weighted fusion method in eq. (3.6), where W_{k-1} and W_{R_k} are the accumulated weight of points in frame $k-1$ and k. Then the weight of each point is updated using eq. (3.7), where W_η is the maximum weight.

$$F_k(\mathbf{p}) = \frac{F_{k-1}(\mathbf{p})W_{k-1}(\mathbf{p}) + F_{R_k}(\mathbf{p})W_{R_k}(\mathbf{p})}{W_{k-1}(\mathbf{p}) + W_{R_k}(\mathbf{p})} \tag{3.6}$$

$$W_k(\mathbf{p}) = \begin{cases} W_{k-1}(\mathbf{p}) + W_{R_k}(\mathbf{p}), & W_k(\mathbf{p}) < W_\eta \\ W_\eta, & W_k(\mathbf{p}) \geq W_\eta \end{cases} \tag{3.7}$$

3.2.3.3 Result of Registration and Reconstruction

The point cloud processing method is programmed using the C++ programming language. To verify the effectiveness of the point cloud registration and reconstruction method described in the former section. This thesis experimented with the process of point cloud acquisition, filtering, denoising, registration and reconstruction in the ROS simulation environment. Figure 3.8(a) shows the simulation environment built in the ROS Gazebo environment. Figure 3.8(b) shows that the camera collected a series of point cloud data around the crab model, which is used as the program's input. Figure 3.8(c) shows the reconstruction result of the crab model. The result meets the expected requirements, and the method and program are validated.

After being validated in the simulation environment, the method is applied to the real experiment environment. As shown in Fig. 3.9(a), the point cloud data of the

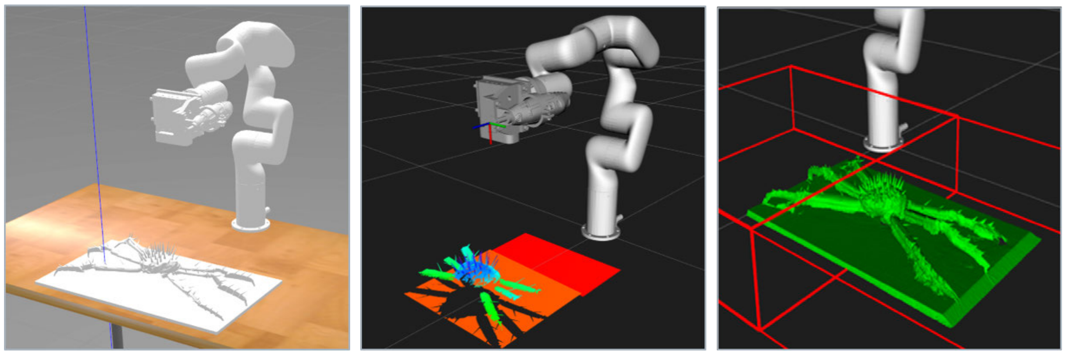

(a) Gazebo simulation setup (b) Point cloud in RViz (c) Simulation result

Figure 3.8: Point cloud processing method validation in ROS simulation environment

crab were captured using a real Azure Kinect camera, and the point cloud data were processed, registered and reconstructed using the same procedure as in the simulation environment.

The spine's shape on the crab shell is a conical shape with small diameters. Due to limitations in the camera's parameters and the principles of depth image acquisition, only a limited number of laser points can be projected onto the spines. Meanwhile, the surface of the spines irregularly reflects the laser light emitted from the camera to other parts of the space. For the above reasons, the laser acquisition sensor on the camera cannot acquire sufficient information about the crab spines. As a result, the features of crab spines could not be well reconstructed. This thesis also tested the latest point cloud processing techniques, such as the BundleFusion [112] and Elastic-Fusion [113], which are all developed based on KinectFusion with the same principle but slight differences. All of them provide almost the same output performance. As shown in Fig. 3.9(b), in addition to the spines' characteristics, the features of the crab shell surface are well reconstructed in the real experiment environment, and this point cloud data is used as the research object to generate the trajectory of robot spine removal.

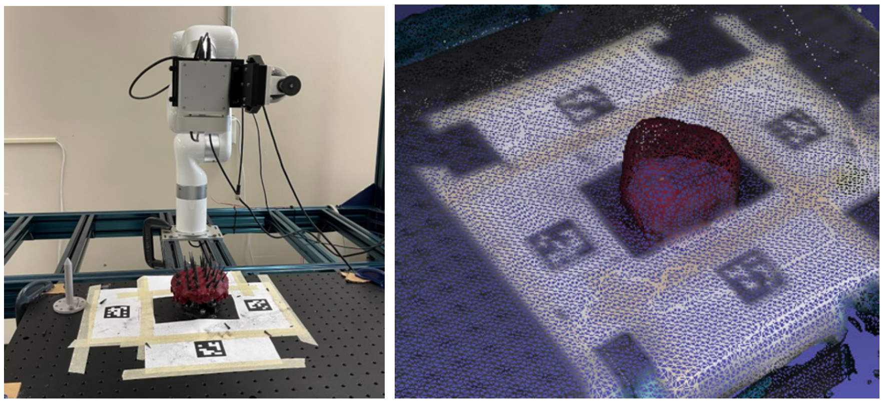

(a) Experiment setup (b) Experiment result

Figure 3.9: Point cloud processing method validation in real experiment environment

3.3 Robot Spine Removal Trajectory Planning

3.3.1 Coordinate System Setup

As shown in Fig. 3.10, four Cartesian coordinate frames are defined: the global frame $\mathcal{B}$ whose origin O_b is fixed at the robot base, a flange coordinate frame $\mathcal{F}$, with its origin O_f fixed at the robot flange, a camera coordinate frame $\mathcal{C}$, whose origin O_c is located at the focal point of the camera, and a robot tool frame $\mathcal{T}$, with its origin O_t. The origin of the robot tool frame is also called the TCP point (tool center point), which is fixed at the tip of the robot end effector.

In this thesis, the robot trajectory for spine removal comprises a set of trajectory points of the TCP point in the global frame $\mathcal{B}$. The pose of each trajectory point can be defined w.r.t. the global frame $\mathcal{B}$.

After processing, the point cloud data will be used as the input to calculate the robot spine removal trajectory. Since the coordinate of the robot trajectory is represented in the global frame $\mathcal{B}$, in order to generate the trajectory, which can be more applicable to the robot, from the point cloud data, the data needs to be transformed into the global frame as well.

Each point in the point cloud data has a unique coordinate, in either the global frame or the moving frame attached to the acquisition device. In this thesis, the crab's point cloud data is obtained by using the high-resolution 3D RGB-D camera, which is installed at the robot end-effector. The raw data is composed of the position information of each point P_i, with coordinate $\mathbf{p}_i^{\mathcal{C}} = [x_i, y_i, z_i]^T$ expressed in the camera frame $\mathcal{C}$. Generating the robot trajectory in the global frame, which is a common practice, involves two coordinate transformations of each point cloud data from the camera frame $\mathcal{C}$ to $\mathcal{B}$.

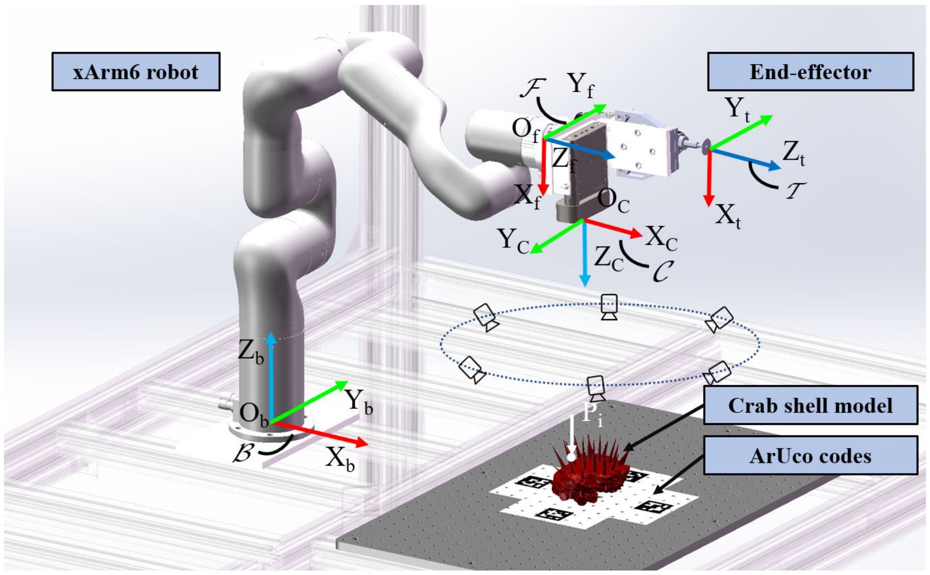


Figure 3.10: Coordinate frame definition on the 6-DOF xArm6 robot

Two coordinate transformations $\mathbf{T}_{\mathcal{C}}^{\mathcal{F}}$ and $\mathbf{T}_{\mathcal{F}}^{\mathcal{B}}$ are involved to express the point cloud data in the robot base frame $\mathcal{B}$, in which $\mathbf{T}_{\mathcal{C}}^{\mathcal{F}}$ transforms the data from $\mathcal{C}$ to $\mathcal{F}$, also known as the hand-eye calibration[3], while $\mathbf{T}_{\mathcal{F}}^{\mathcal{B}}$ further transforms the data from $\mathcal{F}$ to $\mathcal{B}$. Within the framework of the DH parameters [83], $\mathbf{T}_{\mathcal{F}}^{\mathcal{B}}$ can be calculated by means of combining the transformation matrix between two adjacent frames $O_{i-1}X_{i-1}Y_{i-1}Z_{i-1}$ and $O_iX_iY_iZ_i$, which is expressed in eq. (3.8).

$$\mathbf{T}_i^{i-1} = \begin{bmatrix} c(\theta_i) & -s(\theta_i) & 0 & a_{i-1} \\ s(\theta_i)c(\alpha_{i-1}) & c(\theta_i)c(\alpha_{i-1}) & -s(\alpha_{i-1}) & -d_i s(\alpha_{i-1}) \\ s(\theta_i)s(\alpha_{i-1}) & c(\theta_i)s(\alpha_{i-1}) & c(\alpha_{i-1}) & d_i c(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

in which $c(\cdot)$ and $s(\cdot)$ stand for $\cos(\cdot)$ and $\sin(\cdot)$, respectively. As per the modified DH parameters adopted in [83], frame $O_iX_iY_iZ_i$ is attached to link i, where $i = 1, \cdots, 6$. In eq. (3.8), d_i is the offset of X_{i-1} to X_i along Z_i-axis, θ_i being the angle from X_{i-1} to X_i about Z_i-axis, a_i being the length of the common normal of the Z_i axis and Z_{i+1} axis, and α_i denotes the angle from Z_i to Z_{i+1} about the common normal.

By using the DH parameters in Section 2.4, the transformation matrix $\mathbf{T}_{\mathcal{F}}^{\mathcal{B}}$ from frame $\mathcal{F}$ to $\mathcal{B}$ can be expressed as

$$\mathbf{T}_{\mathcal{F}}^{\mathcal{B}} = \mathbf{T}_1^0\mathbf{T}_2^1\mathbf{T}_3^2\mathbf{T}_4^3\mathbf{T}_5^4\mathbf{T}_6^5 \tag{3.9}$$

Thus, any point cloud data, saved in the form of a position vector $\mathbf{p}_i^{\mathcal{C}} = [x_i,\, y_i,\, z_i]$,

[3]Hand-eye (eye-in-hand) calibration is a method to determine the transformation between the robot flange coordinate frame (hand) and the camera coordinate frame (eye) that is mounted on the robot end-effector by using a calibration pattern [114].

is described in the global frame $\mathcal{B}$, namely,

$$\mathbf{p}_i^{\mathcal{B}} = \mathbf{T}_{\mathcal{F}}^{\mathcal{B}} \mathbf{T}_{\mathcal{C}}^{\mathcal{F}} \mathbf{p}_i^{\mathcal{C}} \tag{3.10}$$

3.3.2 Point Cloud Slicing-based Robot Tool Path Planning Method

The main idea of the point cloud slicing method is to create a set of planes E according to a particular requirement as shown in Fig. 3.11(a). These planes intersect with the point cloud data, therefore, a set of cross-section points can be obtained. In addition, The number of the planes, n, can be calculated as

$$n = (y_f - y_l)/d \tag{3.11}$$

where y_f and y_l are the coordinate of the first and the last point along the slicing direction, i.e., along Y-axis, of the pre-processed point cloud data, d being the deburring tool's diameter.

Since the point cloud density is limited, the number of points on the slicing plane E_i may not be sufficient to generate a smooth tool path. In order to address this issue, a common nearest points searching and pairing method should be adopted [115]. As shown in Fig. 3.11(b), two planes E_{il} and E_{ir}, paralleling to the slicing plane E_i, are generated, one on either side, with the same offset distance of $\delta = d/2$ from E_i. The point cloud data sets within the region from E_{il} to E_i, and from E_{ir} to E_i, are defined as K_l and K_r respectively. These data sets are stored by using a $3 \times n$ matrix, with each column representing the position vector of the i^{th} data point P_i^l as $\mathbf{p}_i^l = [x_i^l, y_i^l, z_i^l]$ or P_i^r as $\mathbf{p}_i^r = [x_i^r, y_i^r, z_i^r]$, $i = 0, 1, \ldots, n$. By using the iteration method, all the nearest point pairs P_i^l and P_i^r in the data set of K_l and K_r can be identified. Assuming the slicing direction is along Y-axis, the intersection point of the line formed by the nearest point pairs P_i^l, P_i^r and the plane E_i can be calculated as

$$\begin{cases} x_i^e = (x_i^l - x_i^r)t + x_i^r \\ y_i^e = y_{\min} + (0.5 + j)\delta \\ z_i^e = (z_i^l - z_i^r)t + z_i^r \end{cases} \tag{3.12}$$

where $y_{\min}$ is the minimum y coordinate over all point cloud data points, j representing the sequence number of the slicing plane, and t being defined as

$$t = \frac{x_i^e - x_i^r}{x_i^l - x_i^r} = \frac{y_i^e - y_i^r}{y_i^l - y_i^r} = \frac{z_i^e - z_i^r}{z_i^l - z_i^r} \tag{3.13}$$

Using the PCA method, the intersection point P_i^e is then added into whole point cloud data to calculate the normal vector $\mathbf{n}_{ei}$ of point P_i^e.

The pre-processed point cloud data contains 32,980 points, which is reasonable to achieve an acceptable computational speed for the nearest point pairs searching at this stage. If a camera with higher resolution is used in the future, an octree-based searching algorithm is recommended to be used to obtain the point pairs data in a more efficient way.

After generating the discrete waypoints P_i^e on each slicing plane E_i, they are connected into a whole tool waypoints path $P_{wp,i} = (x_{wp,i}, y_{wp,i}, z_{wp,i})$. Since the slicing direction of each plane is along Y-axis, the path point searching starts from the first intersection point on the first slicing plane. Then, each slicing point along X-axis in this plane will be iterated and connected until there are no more points on

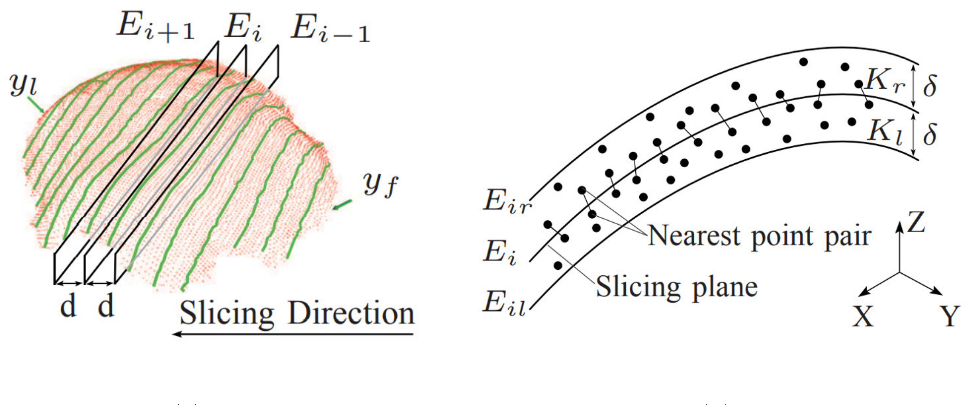


Figure 3.11: Point cloud slicing and nearest point-pair searching

that plane. Following this approach, the search in the inverse direction in continued on the next slicing plane until reaching the end point. Figure 3.12 illustrates the robot tool path generated by searching all points on the selected interaction plane.

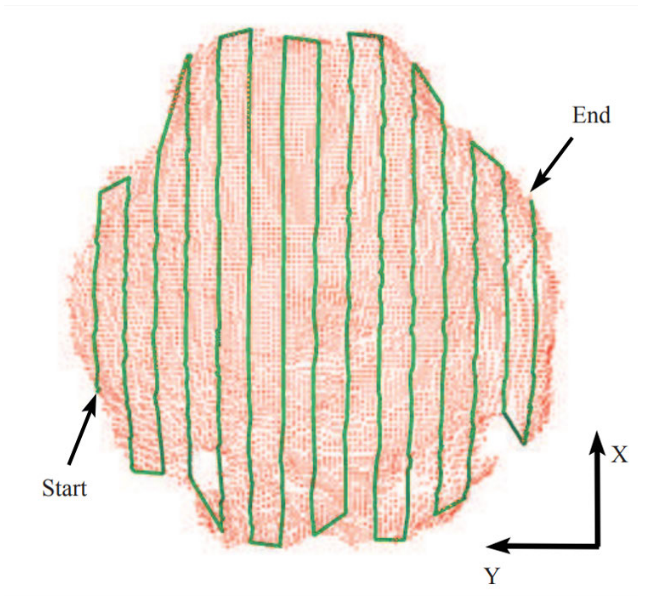


Figure 3.12: The generated robot tool path using point cloud slicing method

3.3.3 Robot Tool Path Planning Method Based on a Combination of 2D Image and 3D Point Cloud

In order to improve the efficiency and accuracy of the point cloud slicing based robot tool path planning method. This section proposes an innovative method incorporating 2D RGB image, 3D depth image and 3D point cloud data to optimize the robot trajectory planning for high accuracy. This method not only allows for the collection of high-resolution images using low-resolution cameras, but also provides more detailed information. The main process of this method is shown in Fig. 3.13.

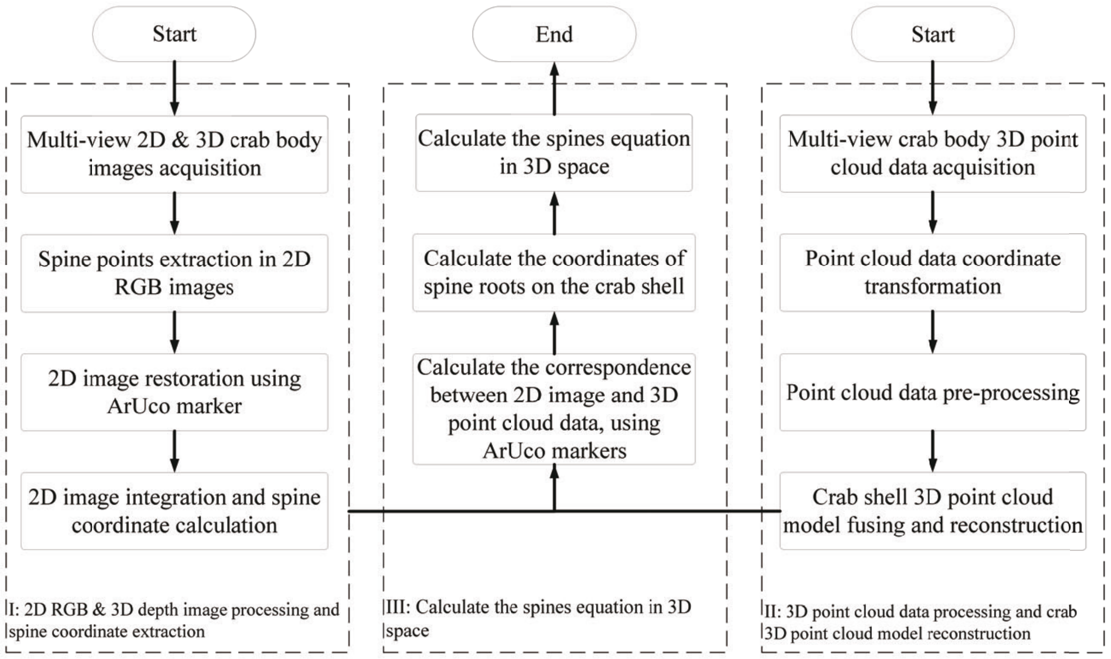

Start
Multi-view 2D & 3D crab body images acquisition
Spine points extraction in 2D RGB images
2D image restoration using ArUco marker
2D image integration and spine coordinate calculation
I: 2D RGB & 3D depth image processing and spine coordinate extraction
End
Calculate the spines equation in 3D space
Calculate the coordinates of spine roots on the crab shell
Calculate the correspondence between 2D image and 3D point cloud data, using ArUco markers
III: Calculate the spines equation in 3D space
Start
Multi-view crab body 3D point cloud data acquisition
Point cloud data coordinate transformation
Point cloud data pre-processing
Crab shell 3D point cloud model fusing and reconstruction
II: 3D point cloud data processing and crab 3D point cloud model reconstruction

Figure 3.13: Robot Tool Path Planning Method based on a Combination of 2D Image and 3D Point Cloud

• **Multi-view 2D RGB Images and 3D Depth Images Capturing**

The Microsoft Azure Kinect camera used in this thesis can capture both 2D RGB and 3D depth images. The robot carries the camera around the crab shell and collects a series of 2D RGB images and 3D depth images, as shown in Fig. 3.14.

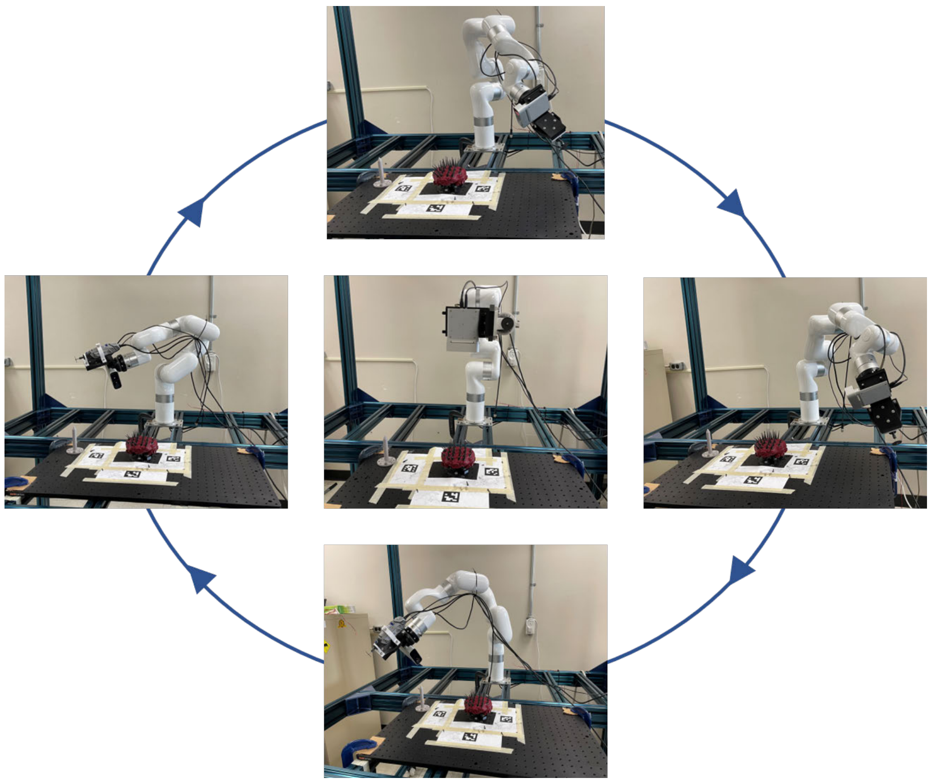


Figure 3.14: Multi-view 2D RGB and 3D depth images capturing

•**Spine Detection and Verification**

This thesis proposes a method to combine the 2D RGB image and 3D depth image to extract information about the crab spines in the 2D RGB image. By processing the 2D RGB image, the edge information of all spines should be extracted first, and then the coordinate of the intersection point of the extracted edge lines can be calculated and used as the potential spines' points. After that, the corresponding points of the potential points in the 3D depth image can be identified and verified on whether the potential point is a point or not. All verified points will be saved for the following process.

Due to environmental interference and other reasons, the 2D image captured by the camera contains noisy information, the most common of which are salt and pepper noise and Gaussian noise [116]. Therefore, this thesis first uses the Gaussian filtering method to remove the noise from the raw image, which is a necessary step for edge detection.

Equation (3.14) represents the two-dimensional Gaussian function [117], where x, y are the pixel coordinates in the Gaussian convolution kernel, and δ is the standard deviation of the Gaussian distribution. In this thesis, a 3×3 Gaussian convolution kernel with a standard deviation of $\delta = 1.5$ is used to process images. This choice was made because the 3×3 kernel size is efficient and provides good results, while larger kernel sizes like 5×5 can cause over-blurring and introduce unwanted side effects. The standard deviation of $\delta = 1.5$ was determined to be the optimal value through experimentation and comparison with other values. Figure 3.15(a) shows the original binary image and Fig. 3.15(b) shows the Gaussian filtered image.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.14}$$

For the next step, the intensity gradient of the filtered image will be calculated by the Sobel [118] edge detection operator. This operator consists of two sets of 3×3 matrices, which are used to convolute with the image to obtain the grayscale values in

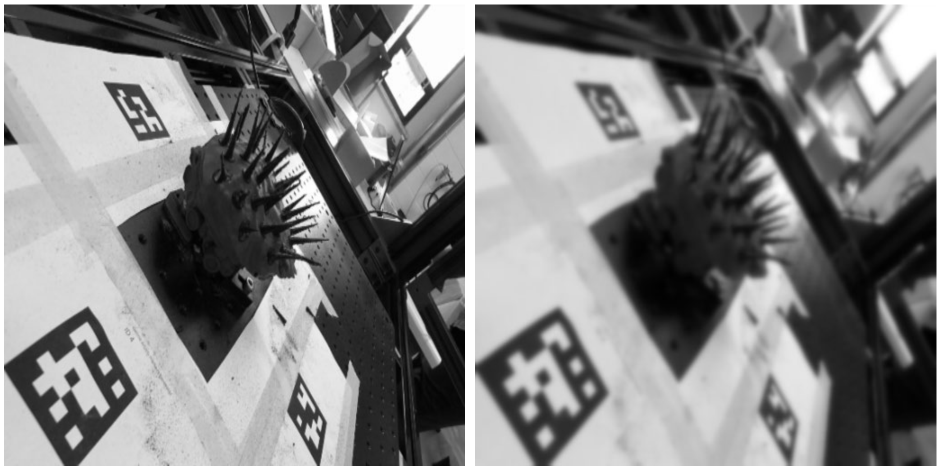

(a) Original image (b) Gaussian filtered image

Figure 3.15: Gaussian 3 × 3 convolution kernel

the horizontal and vertical directions. As shown in eq. (3.15), A represents the filtered image, and G_x, G_y represent the derivative approximations of the filtered image in horizontal and vertical edge detection. Then, the gradient magnitude of each pixel in the filtered image can be calculated using eq. (3.16). As shown in Fig. 3.16, if the gradient magnitude G is higher than the threshold, the pixel will be defined as the edge point. After connecting all the edge points, the edge lines in the filtered image can be extracted, and all the intersection points between edge lines can also be calculated.

$$G_{\mathbf{x}} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} A, G_{\mathbf{y}} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} A \tag{3.15}$$

$$G = \sqrt{G_x^2 + G_y^2} \tag{3.16}$$

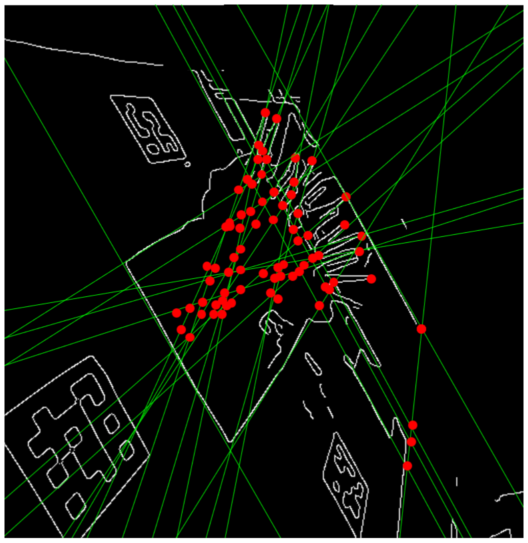
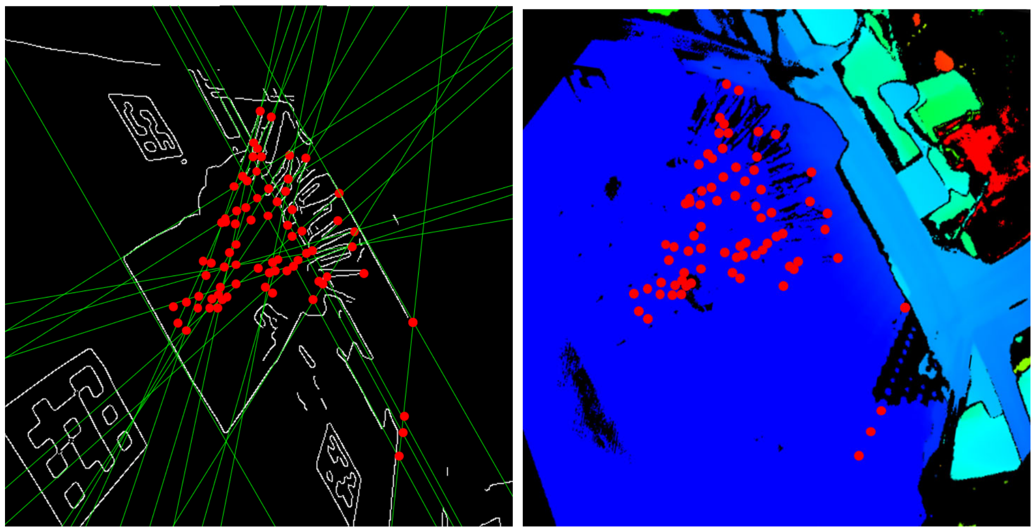


Figure 3.16: Edge and intersection points detection

The coordinate of the intersection points in the 2D image are set as $\mathbf{p}_{i,rgb}(n) = (x_{i,rgb}(n), y_{i,rgb}(n))$, where i is the number of the 2D image, and n is the number of intersection point in the 2D image. As shown in eq. (3.17), the transformation matrix used to aligned the 2D image and 3D depth image can be acquired from the camera's API, where $\mathbf{R}_{r2d}$ and $\mathbf{T}_{r2d}$ are the rotation matrix and translation matrix from the 2D RGB image coordinate system to the 3D depth image coordinate system. $\mathbf{p}_{i,depth}(n) = (x_{i,depth}(n), y_{i,depth}(n), z_{i,depth}(n))$ represents the calculated corresponding point of $\mathbf{p}_{i,rgb}(n)$ in the 3D depth image coordinate system. Figure 3.17 shows the corresponding point pairs in the 2D RGB and 3D depth image.

$$\mathbf{p}_{i,depth}(n) = \mathbf{R}_{r2d}\mathbf{p}_{i,rgb}(n) + \mathbf{T}_{r2d} \tag{3.17}$$

After calculating the corresponding point $\mathbf{p}_{i,depth}(n)$ in the 3D depth image, the next step is to determine whether these points are the point of the crab spines or not.



(a) Image with edge and intersection points (b) Depth image with corresponding points

Figure 3.17: Corresponding point pairs in the 2D and 3D image

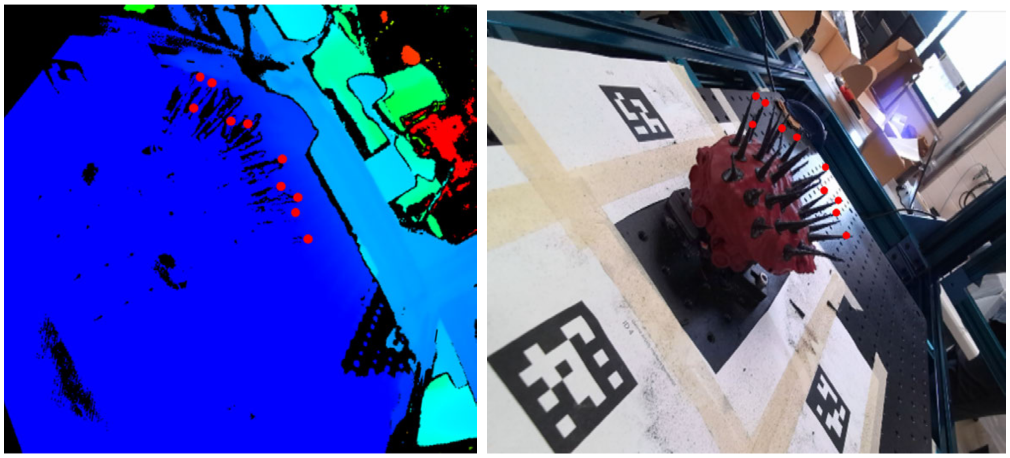
As shown in eq. (3.18), this thesis proposes a method of calculating the depth mean value $\bar{z}$ of the t nearest points around point $\mathbf{p}_{i,depth}(n)$. As shown in Fig. 3.18, if the depth mean value is between the threshold of z_0 and z_1, the point is considered as the point of the crab spine, and the corresponding point back in the 2D image is stored into another array for further processing.

$$\bar{z} = \frac{\sum_{i=1}^{t} z_i}{t} \tag{3.18}$$

•Spine Points Transformation Using ArUco Marker

In the previous section, the coordinates of points of crab spines in the 2D image at the different shooting angles have been extracted. The next step is to convert these coordinates to the XY plane in the global coordinate system using the ArUco marker.

The ArUco marker [75] is a binary square marker that can be used for camera pose estimation, which consists of a wide black boundary and an internal binary matrix



(a) Processed spine points in 3D image (b) Processed spine points in 2D image

Figure 3.18: Processed spine points in 2D and 3D images

pattern. The binary matrix pattern contains the ID information of the marker, and the black boundary can be used to extract the boundary and corner point information accurately. The projection relationship between the ArUco marker and the camera can be calculated by combining the matrix pattern with the boundary information. As shown in Fig. 3.19, four ArUco markers are pasted around the crab model. The boundary, marker number, and marker coordinate are identified and displayed.

This thesis uses ArUco maker to identify the projection relationship of the images from a tilted view and a XY plane within the global coordinate system. Equation (3.19) shows the perspective transformation matrix that is read from the ArUco API. As shown in Fig. 3.19, using this matrix, the spine points in the tilted image can be transformed into the XY plane within the global coordinate system, where $\mathbf{p}_{i,h}(x_h, y_h)$ is the spine point coordinate in the XY plane of the global coordinate system in the number i image.

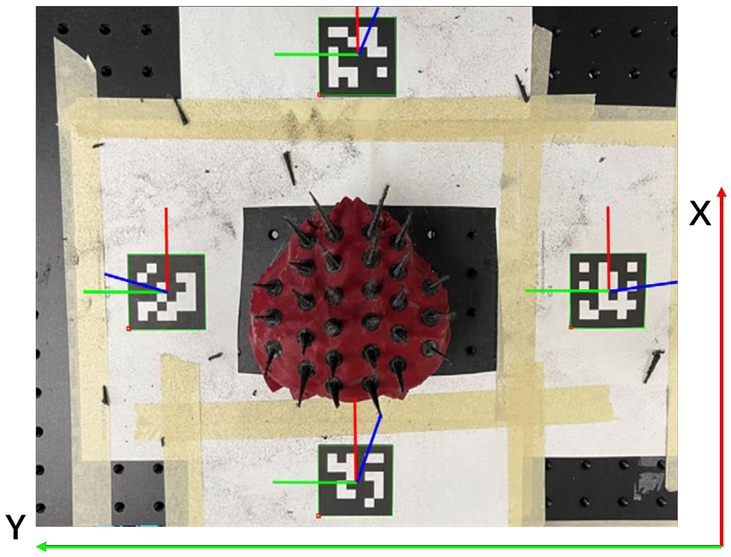


Figure 3.19: ArUco markers setting up and identification

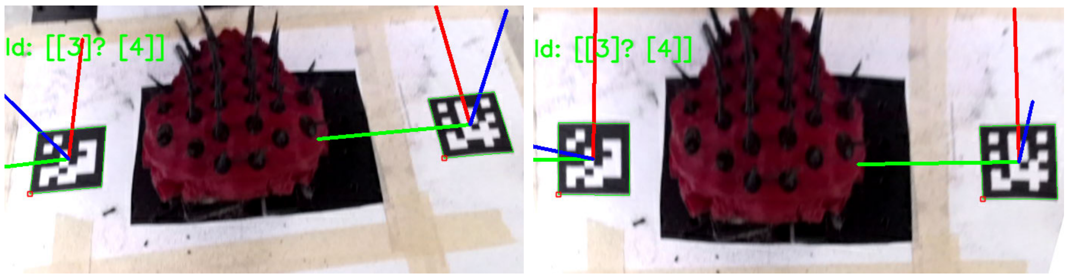
$$\mathbf{M}_t^h = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \tag{3.19}$$

$$\mathbf{p}_{i,h}(x_h, y_h) = \mathbf{p}_{i,rgb,c}(x_t, y_t)\mathbf{M}_t^h = [\frac{M_{11}x_t + M_{12}y_t + M_{13}}{M_{31}x_t + M_{32}y_t + M_{33}}, \frac{M_{21}x_t + M_{22}y_t + M_{23}}{M_{31}x_t + M_{32}y_t + M_{33}}]^T \tag{3.20}$$

•Image Integration and Spine Coordinate Calculation

After the coordinates of the spine points in the 2D RGB image of all shooting perspectives have been converted to the XY plane within the global coordinate system, all the converted spine points will be aggregated into an array as shown below, where s is the number of the spine point:

$$\mathbf{p}_d(s) = \{\mathbf{p}_{1,h}(n), \mathbf{p}_{2,h}(n), \mathbf{p}_{3,h}(n), \ldots, \mathbf{p}_{i,h}(n)\} \tag{3.21}$$



(a) Before restored image (b) After restored image

Figure 3.20: Restored image with spine points by using ArUco code

All the spine points on the crab shell in $\mathbf{p}_d(s)$ are shown in Fig. 3.21, the results show that each spine corresponds to a series of discrete points in the X-Y plane due to the random error during the image restoration method in the former section. The k-means clustering method [119] is chosen to classify all the discrete points into several groups to extract the spine point coordinate. Equation (3.22) shows the equation of k-means clustering method, where $\mathbf{p}_d(i)$ is the converted spine points in the former section, k is the maximum number of clusters, s is the total number of discrete points, and $\mathbf{c}_{spine,2d}(j) = (x_{spine,2d}(j), y_{spine,2d}(j))$ is the centroid coordinate of the group j. As shown in Fig. 3.22, when the points in different groups are at the minimum distance from the centroid, the categorizing method is complete. The centroid $\mathbf{c}_{spine,2d}(j)$ of each point group can be used as the projection point of the crab spines on the X-Y plane within the global frame $\mathcal{B}$.

$$E = \sum_{j=1}^{k} \sum_{i=1}^{s} \|\mathbf{p}_d(i) - \mathbf{c}_{spine,2d}(j)\|^2 \tag{3.22}$$

So far, the spine points of the crab have been extracted in the 2D image. In Fig. 3.21, center points of four ArUco markers $\mathbf{c}_{Aru,2d}(1), \mathbf{c}_{Aru,2d}(2), \mathbf{c}_{Aru,2d}(3), \mathbf{c}_{Aru,2d}(4)$ in the X-Y plane within the global coordinate frame $\mathcal{B}$ are read from the OpenCV

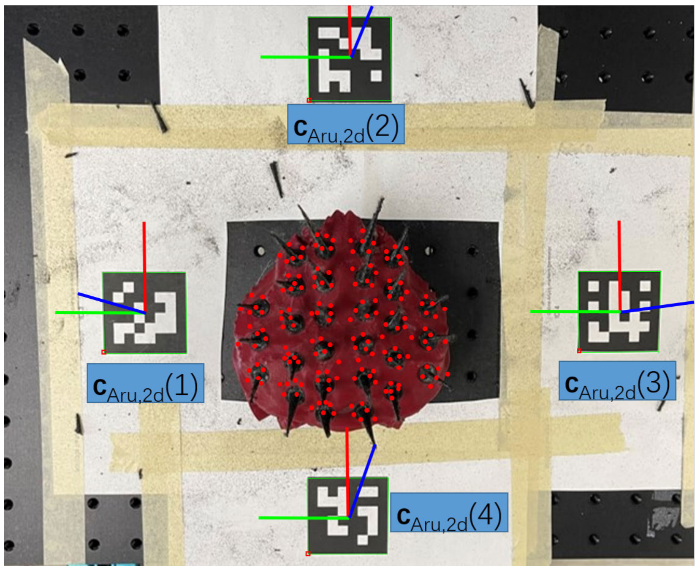


Figure 3.21: Discrete spine points on the X-Y plane in global coordinate frame.

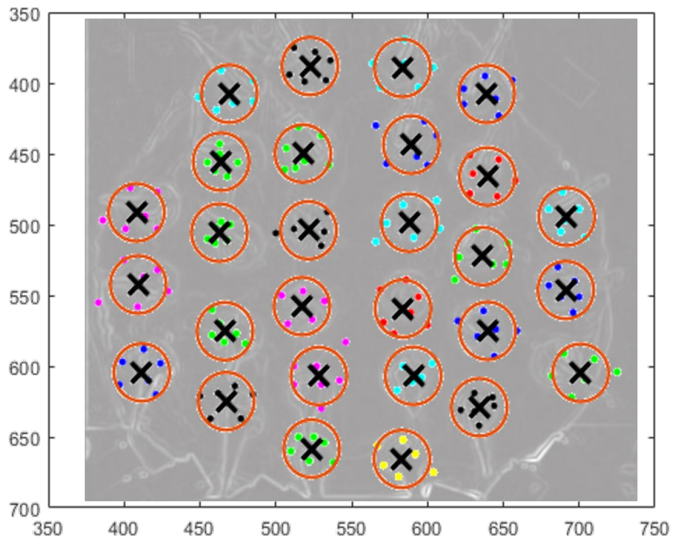


Figure 3.22: Discrete spine points categorizing using k-means clustering method.

ArUco API. The following sections will introduce the processing method of the 3D point cloud data of the crab.

•Spines feature calculation in 3D point cloud model

An ArUco marker based method is developed in this section to find the correlation between the 2D image in the XY plane and the 3D point cloud data in the global coordinate system that is reconstructed in Section 3.2.3.3. The format of the reconstructed point cloud model in Section 3.2.3.3 is XYZRGB, where XYZ represents the position coordinates and RGB represents the color of each point.

As shown in Fig. 3.23(a) and (b), the coordinates of the center points of the four ArUco markers $\mathbf{c}_{Aru,2d}(1)$, $\mathbf{c}_{Aru,2d}(2)$, $\mathbf{c}_{Aru,2d}(3)$, $\mathbf{c}_{Aru,2d}(4)$ in the top view of the 2D RGB image and $\mathbf{c}_{Aru,3d}(1)$, $\mathbf{c}_{Aru,3d}(2)$, $\mathbf{c}_{Aru,3d}(3)$, $\mathbf{c}_{Aru,3d}(4)$ in the ArUco marker plane in the 3D point cloud data have been calculated in the previous sections. Then by using ArUco markers as a medium, the relationship between 2D image and 3D point cloud data can be established, and the corresponding spine projection points on the ArUco marker plane in the 3D point cloud data can be calculated.

The corresponding spine projection points $\mathbf{c}_{spine,3d}(j) = (x_{spine,3d}(j), y_{spine,3d}(j), z_{spine,3d}(j))$ on the Aruco marker plane in the 3D point cloud data with the point $\mathbf{c}_{spine,2d}(j)$ in the 2D RGB image can be calculated by using eq. (3.23), where k_x and k_y are the scale factors of the 2D image to the ArUco marker plane in the 3D point cloud data, A, B, C and D are the coefficients of the ArUco marker plane. As shown in Fig. 3.23(c), red dots represent the calculated corresponding spine projection points on the ArUco marker plane in the point cloud data.

$$\mathbf{c}_{spine,3d}(j) = \begin{bmatrix} x_{spine,3d}(j) \\ y_{spine,3d}(j) \\ z_{spine,3d}(j) \end{bmatrix}$$

$$= \begin{bmatrix} x_{aru,3d}(2) - k_x(x_{aru,2d}(2) - x_{spine,2d}(j)) \\ y_{aru,3d}(1) - k_y(y_{aru,2d}(1) - y_{spine,2d}(j)) \\ -(Ax_{spine,3d}(j) + By_{spine,3d}(j) + D)/C \end{bmatrix} \tag{3.23}$$

$$k_x = \frac{x_{aru,3d}(2) - x_{aru,3d}(4)}{x_{aru,2d}(2) - x_{aru,2d}(4)}$$

$$k_y = \frac{y_{aru,3d}(1) - y_{aru,3d}(3)}{y_{aru,2d}(1) - y_{aru,2d}(3)}$$

In order to restore the spine's features, the spine projection points on the ArUco marker plane need to be projected back to the crab shell surface. The equations of the line $L(j)$ passing through each projection point $\mathbf{c}_{spine,3d}(j)$ and being perpendicular to the ArUco marker plane can be calculated by eq. (3.24), where $[A, B, C]^T$ are the normal vector coordinates to the plane, t is the line's scalar parameter.

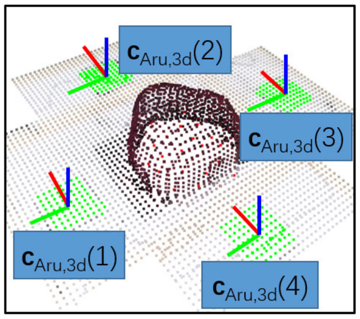
$$L(j) \leftarrow \begin{cases} x = x_{spine,3d}(j) + At \\ y = y_{spine,3d}(j) + Bt \\ z = z_{spine,3d}(j) + Ct \end{cases} , (-\infty < t < +\infty) \tag{3.24}$$

By using the nearest point searching method, the distance between each point $\mathbf{p}_i$ in the point cloud data to the line $L(j)$ can be calculated by eq. (3.25). The point with the minimum distance for each line is defined as the spine root point $\mathbf{p}_{rt}(j)(x_{rt}(j),$ $y_{rt}(j)$,$z_{rt}(j))$ on the crab shell, which are shown by the green dots in Fig. 3.23(d).

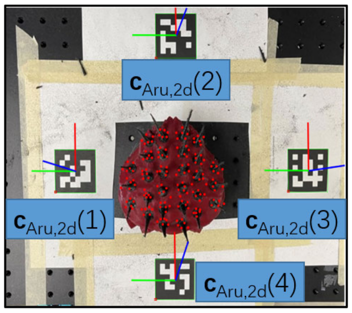
$$\begin{aligned}
&d_i = \sqrt{\Delta x + \Delta y + \Delta z} \\
&\Delta x = (x_i - x_{spine,3d}(j) - At_i{}')^2 \\
&\Delta y = (y_i - y_{spine,3d}(j) - Bt_i{}')^2 \\
&\Delta z = (z_i - Z_{spine,3d}(j) - Ct_i{}')^2 \\
&t_i{}' = \\
&\frac{A(x_i - x_{spine,3d}(j)) + B(y_i - y_{spine,3d}(j)) + C(z_i - z_{spine,3d}(j))}{A^2 + B^2 + C^2}
\end{aligned} \tag{3.25}$$

The last step is to calculate the equation of the spine lines, which are passing through the spine root point $\mathbf{p}_{rt}(j)$ and perpendicular to the crab's outer surface. The normal vector of the spine root point on the crab shell can be calculated as $\mathbf{n}(j) = [n_x(j), n_y(j), n_z(j)]^T$. Therefore, the spine lines equation can be calculated by eq. (3.26). As shown in Fig. 3.23(e), all the spine lines are calculated and visualized as the yellow lines.

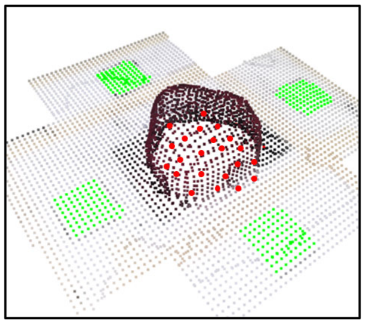
$$\frac{x - x_{rt}(j)}{n_x(j)} = \frac{y - y_{rt}(j)}{n_y(j)} = \frac{z - z_{rt}(j)}{n_z(j)} \tag{3.26}$$



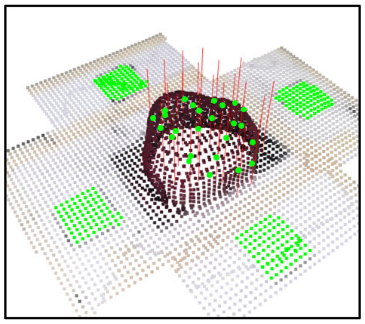
(a)



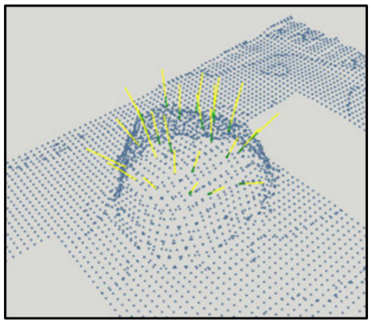
(b)



(c)



(d)



(e)

Figure 3.23: 2D image and 3D point cloud correspondence and spines calculation: (a) ArUco markers center points in the 3D point cloud data; (b) ArUco markers center points in the 2D image; (c) Spine projection points on the ArUco marker plane; (d) Spine root points on the crab shell; (e) Spine lines on the crab shell.

•Spine removal tool path for single spine

A disc-shaped fibreglass-reinforced cut-off wheel is chosen as the cutting tool. To prevent the cutting tool from cutting into the crab shell and to ensure the efficiency, the cutting tool path must remain perpendicular to the spines while cutting each spine and the cutting path of the spine is offset by a certain distance relative to the spine root point $\mathbf{p}_{rt}(j)$ in the spine growth direction.

As shown in Fig. 3.24, the equation of plane $\alpha(j)$ passing through the spine root point $\mathbf{p}_{rt}(j)$ on the crab outer surface and perpendicular to the spine growth direction can be calculated by eq. (3.27), where $n_x(j), n_y(j), n_z(j)$ are the values of the normal vector on the spine root point, $A_\alpha, B_\alpha, C_\alpha, D_\alpha$ are the coefficients of the plane $\alpha(j)$.

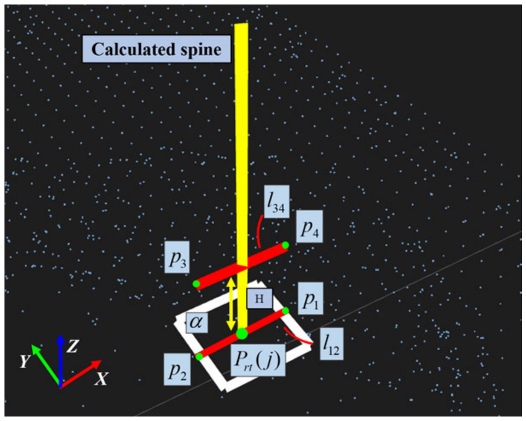


Figure 3.24: Cutting tool path calculation.

$$
\begin{aligned}
&A_\alpha x + B_\alpha y + C_\alpha z + D_\alpha = 0 \\
&n_x(j)x + n_y(j)y + n_z(j)z + D_\alpha = 0 \\
&D_\alpha = (-1)[n_x(j)x_{rt}(j) + n_y(j)y_{rt}(j) + n_z(j)z_{rt}(j)]
\end{aligned}
\tag{3.27}
$$

The point P_1 and P_2 lie on the plane $\alpha(j)$, and the line l_{12} formed by P_1 and P_2 passes through the spine root point $\mathbf{p}_{rt}(j)$ and perpendicular to the spine line. P_1 and P_2 are 5 mm away from the point $\mathbf{p}_{rt}(j)$ along the positive and negative direction of the X-axis, respectively. The coordinates of the points P_1 and P_2 can be calculated by eqs. (3.28) and (3.29), as follows

$$\mathbf{p}_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} x_{rt}(j) + 0.005 \\ y_{rt}(j) \\ (-1)[D_\alpha + n_x(j)x_1 + n_y(j)y_1]/n_z(j) \end{bmatrix} \tag{3.28}$$

$$\mathbf{p}_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x_{rt}(j) - 0.005 \\ y_{rt}(j) \\ (-1)[D_\alpha + n_x(j)x_2 + n_y(j)y_2]/n_z(j) \end{bmatrix} \tag{3.29}$$

The line l_{34}, consisting of point P_3 and P_4, is parallel to the line l_{12} and with an offset distance, $H = 5$ mm along the spine growth direction. The coordinate of point P_3 and P_4 can be calculated by eqs. (3.30)–(3.32). The line l_{34} is defined by eq. (3.33). The line segment between P_3 to P_4 will be used as the cutting tool path for the spine.

$$\mathbf{p}_3 = \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_2 + n_x(j)k \\ y_2 + n_y(j)k \\ z_2 + n_z(j)k \end{bmatrix} \tag{3.30}$$

$$\mathbf{p}_4 = \begin{bmatrix} x_4 \\ y_4 \\ z_4 \end{bmatrix} = \begin{bmatrix} x_1 + n_x(j)k \\ y_1 + n_y(j)k \\ z_1 + n_z(j)k \end{bmatrix} \tag{3.31}$$

$$\begin{aligned} H &= \sqrt{(x_4 - x_1)^2 + (y_4 - y_1)^2 + (z_4 - z_1)^2} \\ H &= k\sqrt{n_x(j)^2 + n_y(j)^2 + n_z(j)^2} \\ k &= \frac{H}{\sqrt{n_x(j)^2 + n_y(j)^2 + n_z(j)^2}} \end{aligned} \tag{3.32}$$

$$l_{34} : \frac{x - x_3}{x_4 - x_3} = \frac{y - y_3}{y_4 - y_3} = \frac{z - z_3}{z_4 - z_3} \tag{3.33}$$

3.3.4 Robot Tool Pose Calculation

In this section, the method of estimating the pose of the robot tool along each waypoint on the robot tool path is explained.

As shown in Fig. 3.3, the normal vector $\mathbf{n}_i = [n_{x_i}, n_{y_i}, n_{z_i}]$ of the i^{th} waypoint is expressed in the robot base frame $\mathcal{B}$. Robot pose in ROS is represented by using quaternion. Quaternion, an elegant, straightforward and computationally robust tool for pose reprentation, provides a compact description without the drawbacks in terms of other representations, e.g., the gimbal lock brought by Euler angles. Quaternion consists of four parameters, namely, one scalar and one three-dimensional vector [120]. In this thesis, the normal vector of the waypoint is transformed into the Euler angles with extrinsic rotational sequence along the fixed axes. The Euler angle representation is then transformed into quaternion.

Assuming that α, β and γ are the rotation angles of the robot end effector around x-, y- and z-axis, respectively. They are calcuated from the normal vector of the i^{th} waypoint, $\mathbf{n}_i$, as

$$\begin{cases} \alpha = -\text{atan2}(n_{y_i}, n_{z_i}) - \pi \\ \beta = \text{atan2}(n_{x_i}, n_{z_i}) \\ \gamma = 0 \end{cases} \tag{3.34}$$

Note that in terms of rotating the tool direction around x- and y-axis by a certain angle, it can always be colinear with $\mathbf{n}_i$, i.e., γ can be set to 0 without rotating around z-axis. The quaternion $\boldsymbol{\eta_{wp,i}}$ of the tool pose at the i^{th} waypoint can be calculated as follows

$$\boldsymbol{\eta_{wp,i}} = \begin{bmatrix} qw_{wp,i} \\ qx_{wp,i} \\ qy_{wp,i} \\ qz_{wp,i} \end{bmatrix} = \begin{bmatrix} c(\alpha/2)c(\beta/2)c(\gamma/2) + s(\alpha/2)s(\beta/2)s(\gamma/2) \\ s(\alpha/2)c(\beta/2)c(\gamma/2) - c(\alpha/2)s(\beta/2)s(\gamma/2) \\ c(\alpha/2)s(\beta/2)c(\gamma/2) + s(\alpha/2)c(\beta/2)s(\gamma/2) \\ c(\alpha/2)c(\beta/2)s(\gamma/2) - s(\alpha/2)s(\beta/2)c(\gamma/2) \end{bmatrix} \tag{3.35}$$

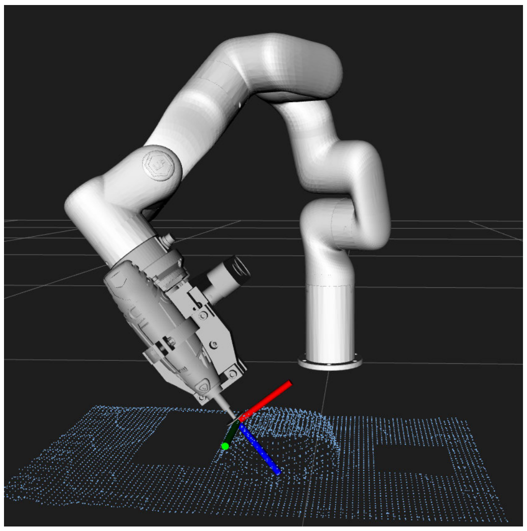


Figure 3.25: Simulation of robot tool pose

Figure 3.25 shows the simulation results of the pose of the robot end-effector, and the red, green and blue bars indicate the x-, y- and z-axis of the local coordinate system attached to the robot tool, in which x-axis is parallel to the camera optical axis, and z-axis is parallel to the normal vector of the waypoint.

3.3.5 Complete Spine Removal Tool Path Generation and Optimization

After the spine removal tool path for each spine has been calculated, the next step is to merge them into a complete spine removal tool path for the whole crab shell. As shown in Fig. 3.26(a), directly connecting the spine removal tool path using straight line segments method would lead to unexpected results, such as discontinuous speed and acceleration of the robot, which will impose negative impact on the cutting performance and efficiency. Therefore, the cubic spline interpolation method [121] is chosen to generate a smooth and optimized robot tool path.

The start point coordinate of the current path segment is $\mathbf{p}_k$. The start time and start velocity are set to be t_k and $\mathbf{v}_k$. The endpoint coordinate of the current path segment is denoted by $\mathbf{p}_{k+1}$. The end time and the end velocity are set to be t_{k+1} and $\mathbf{v}_{k+1}$, respectively, which are assumed known. The cubic spline interpolation method is described by eq. (3.36). Substituting $\mathbf{p}_k$, $\mathbf{v}_k$, $\mathbf{p}_{k+1}$ and $\mathbf{v}_{k+1}$ into eq. (3.36) leads to eq. (3.37), in which $T_k = t_{k+1} - t_k$ is the execution time of the path segment. The coefficients a_0, a_1, a_2 and a_3 can be calculated by using eq. (3.38). The equation of the cubic spline curve can be obtained by substituting the optimal cutting speed verified by the manual experiment into eqs. (3.36) and (3.38). The xyz coordinate of the point on the cubic spline curve can be thus calculated, which will be used as the smoothed tool path for the robot.

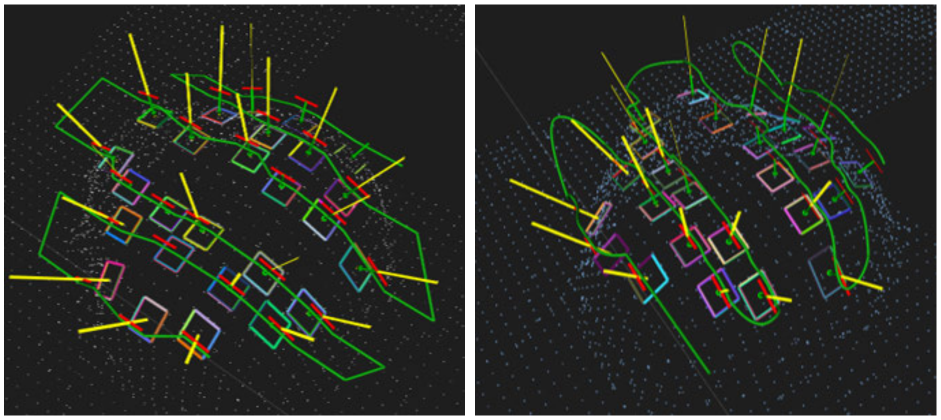
$$\mathbf{p}_k(t) = a_0 + a_1 (t - t_k) + a_2 (t - t_k)^2 + a_3 (t - t_k)^3 \tag{3.36}$$

$$\begin{cases} \mathbf{p}_k(t_k) = a_0 & = \mathbf{p}_k \\ \dot{\mathbf{p}}_k(t_k) = a_1 & = \mathbf{v}_k \\ \mathbf{p}_k(t_{k+1}) = a_0 + a_1 T_k + a_2 T_k^2 + a_3 T_k^3 & = \mathbf{p}_{k+1} \\ \dot{\mathbf{p}}_k(t_{k+1}) = a_1 + 2a_2 T_k + 3a_3 T_k^2 & = \mathbf{v}_{k+1} \end{cases} \tag{3.37}$$

$$\begin{cases} a_0 = \mathbf{p}_k \\ a_1 = \mathbf{v}_k \\ a_2 = \dfrac{1}{T_k}\left[\dfrac{3(\mathbf{p}_{k+1} - \mathbf{p}_k)}{T_k} - 2\mathbf{v}_k - \mathbf{v}_{k+1}\right] \\ a_3 = \dfrac{1}{T_k^2}\left[\dfrac{2(\mathbf{p}_k - \mathbf{p}_{k+1})}{T_k} + \mathbf{v}_k + \mathbf{v}_{k+1}\right] \end{cases} \tag{3.38}$$

In additional to calculating the position of the point on the smoothed tool path, the pose of the robot at each point also needs to be calculated. The spherical linear interpolation(SLERP) method [122] is used for calculation. As shown in eq. (3.39), where η_m is the interpolated quaternion, η_a and η_b are the two quaternions to be interpolated, t is a scalar between 0 (at η_a) and 1 (at η_b), θ is the angle between η_a and η_b, and $t\theta$ is the angle between η_m and η_b. The tool path after optimizing and smoothing is shown in Fig. 3.26(b), which shows the robot tool path after optimization is much smoother.

$$\eta_{\mathbf{m}} = \mathrm{Slerp}(\eta_{\mathbf{a}}, \eta_{\mathbf{b}}, t) = \frac{\sin[(1-t)\theta]\eta_{\mathbf{a}} + \sin(t\theta)\eta_{\mathbf{b}}}{\sin\theta} \tag{3.39}$$



(a) (b)

Figure 3.26: Tool path generation and optimization (a) Before optimization (b) After optimization.

3.4 Summary

This chapter described the core methodology. Firstly, point cloud filtering, de-noising and surface normal calculation methods are introduced. Moreover, the TSDF-based point cloud registration and reconstruction methods are discussed in detail. After that, the novel point cloud slicing-based tool path planning method following with the combination of 2D Image and 3D point cloud robot tool path planning methods are introduced.

Furthermore, the robot pose calculation method at each point on the robot tool path is proposed, and a cubic spline interpolation method combined with the spherical linear interpolation method is used to optimize the robot tool path.

Chapter 4

Simulation of the Robot Spine Removal System

This section builds an experiment system in the ROS RViz simulation environment to verify the proposed porcupine crab spines removal method. The 3D model of the real xArm6 six-axis robot and the end-effector are loaded into the simulation environment. The porcupine crab's 2D image and 3D point cloud data are acquired using the Microsoft Azure Kinect RGB-D camera and then processed by the algorithm. The results of the point cloud processing and the robot tool path generation are visualized and verified in the simulation environment.

4.1 ROS-based Simulation Environment Setup

In order to achieve an expected simulation performance, the robot, the end–effector and accessories are loaded into the ROS RViz simulation environment with the same dimensions and mounting positions as their counterparts in the real world. Figure 4.1

shows the robot platform. The model of the cutting tool is Dremel 3000 rotatry tool with a maximum rotational speed of 35,000 rpm. A disc-shaped fiberglass cutoff wheel is installed into the cutting tool. The model of the camera is Microsoft Azure Kinect. The coordinate system is established in the simulation environment as described in Section 3.3.1.

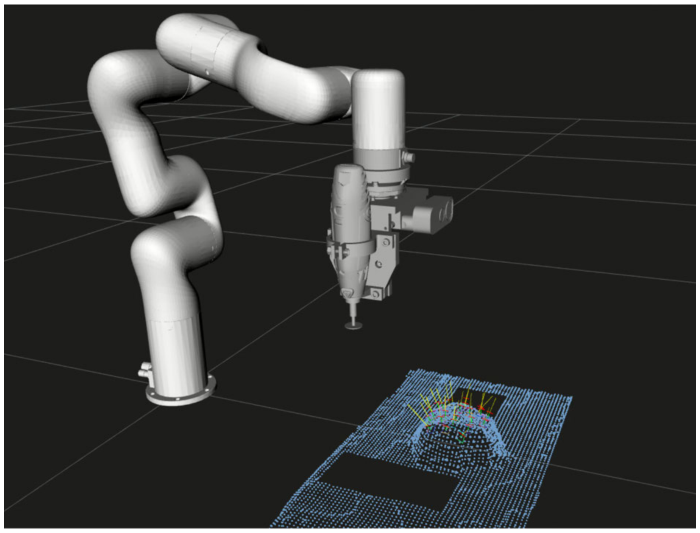


Figure 4.1: Robot platform in ROS RViz simulation environment

4.2 Simulation Result

4.2.1 Point Cloud Slicing-based Robot Tool Path Planning Method Simulation Result

A simulation software package within the framework of ROS RViz is developed using C++, to validate the proposed method. The 3D point cloud data of the crab model is captured using the Microsoft Azure kinect RGB-D camera, which is further saved as a file with the PCD format. As shown in Fig. 4.2, the raw point cloud data is noise-contaminated and also contain other unnecessary feature points. After filtering and denoising, the point cloud data that contains essential information of the crab shell is well extracted. The normal vector of each surface point need to be generated as well, of which directions are highlighted with green arrows.

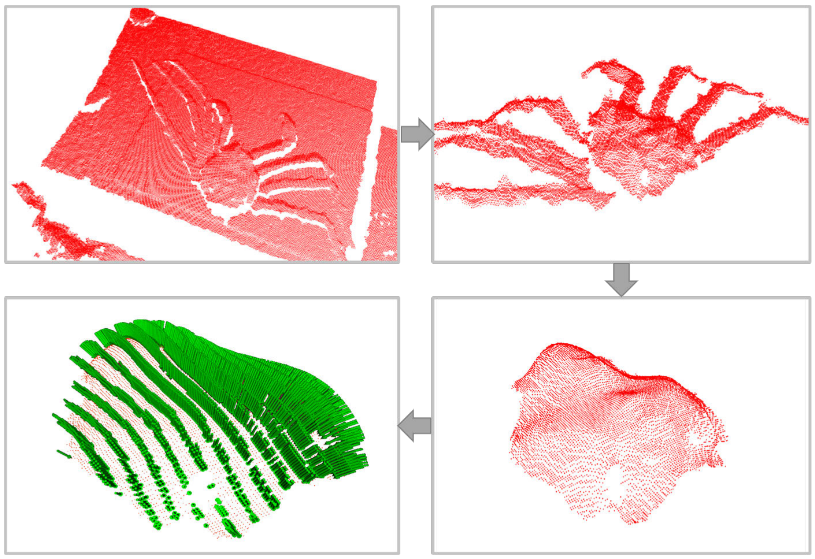


Figure 4.2: Point cloud data processing and surface normal calculation

Figure 4.3 shows the result of the point cloud slicing and tool path generation method in ROS RViz. The green curves represent the robot tool path, the origin of the robot tool frame being fixed at the tip of the robot tool. It is noteworthy that the z-axis of the tool frame is collinear with the normal vector of the tool path. By using the TRAC-IK inverse kinematic solver, robot joint parameters corresponding to all points on the robot tool path can be calculated within 15 seconds, which indicates that the robot system has a good reachability. The results reveal that the tool path generation method is effective and efficient to process complex shapes such as crab shells, and the design of the robot end effector is appropriate.

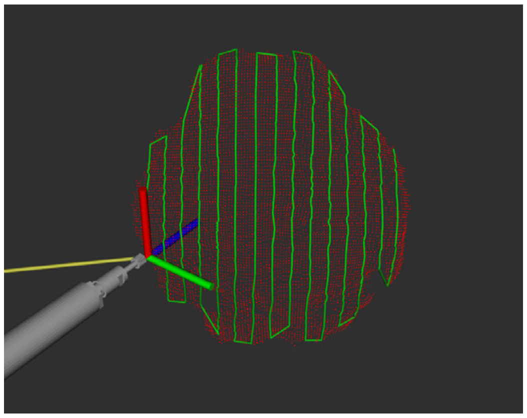
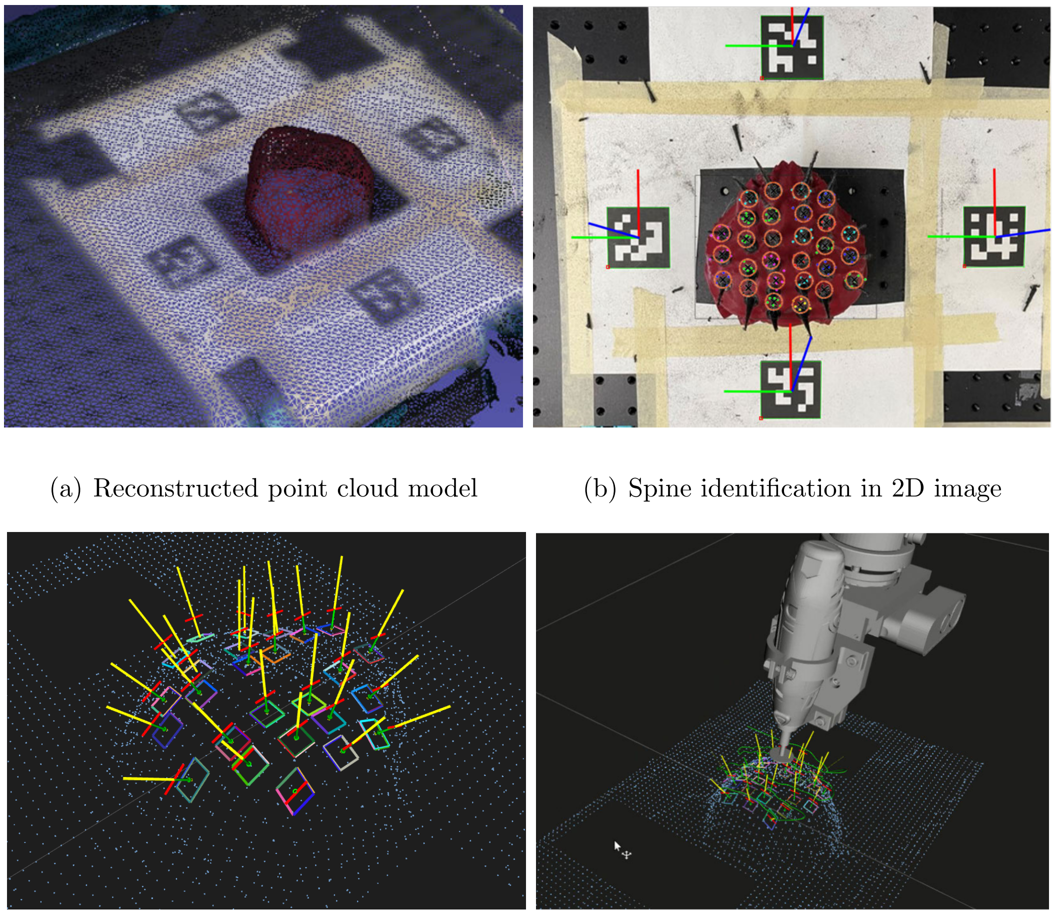


Figure 4.3: Simulation result of the first tool path planning method

4.2.2 Result of the Robot Tool Path Planning Method Based on a Combination of 2D Image and 3D Point Cloud

The simulation of this method is performed in the same testing environment as the previous method. Around the crab body, a series of 2D RGB images and 3D point cloud data are captured by using a real Microsoft Azure Kinect RGB-D camera and processed as described in Section 3.3.3. As shown in Fig. 4.4(a), coloured point cloud data that contains features of the whole crab body is reconstructed. As shown in Fig. 4.4(b), the spines in the multi-angle images are identified and extracted, and the pixel coordinate of each spine root point is marked. As shown in Fig. 4.4(c), using the ArUco code as a medium, the relationship between the 2D image and 3D point cloud data can be established, and the position and growth direction of the spines on the crab shell in the point cloud data can be calculated. Figure 4.4(d) shows the result of the 2D image and 3D point cloud data combination robot tool path planning method. The green curves represent the robot tool path, and the z-axis of the tool frame is collinear with the normal vector of the tool path. The results show that the robot tool path generated by this method can enhance the smoothness of the path, and the total length of the tool path is shorter than the previous method, which can improve the efficiency of the spines removal process.


(a) Reconstructed point cloud model (b) Spine identification in 2D image

(c) Processed spines in 3D point cloud data (d) Generated robot tool path

Figure 4.4: Simulation result of the 2D image and 3D point cloud method

4.2.3 Summary

This section establishes the simulation platform as its real counterpart in the ROS simulation environment. Two tool path planning methods are tested and visualized based on the 2D image and 3D point cloud data collected by the real camera. The simulation results show that both methods can meet the tool path planning requirements and generate a complete and feasible robot tool path without any unreachable points. Meanwhile, the robot's poses along the tool path are well adapted to the surface curvature and can be compliant to the surface curvature changes of the crab shell. Collision detection is performed during the simulation, and no collision error has been observed. The feasibility of the path planning algorithm can be verified by the simulation results. By comparing with the previous method, the tool path generated by the 2D image and 3D point cloud combination method is more effective and efficient, and will be adapted for experimental verification in the following section.

Chapter 5

Experiment of the Robot Spine Removal System

In this section, the experiment platform in the real environment is built to conduct the whole process of robot spine removal. Firstly, the robot tool frame is calibrated to determine the relationship between the tool frame to the global/world frame. Then the 2D RGB image camera and the 3D depth camera, i.e., the Microsoft Azure Kinect camera, are calibrated as well to determine the relationship between the camera and the global/world frames by using the hand-eye calibration method. Moreover, the intrinsic matrix and the distortion parameters should be calculated to improve the camera's overall accuracy. After that, the real robot spine removal experiment, using the 2D image and 3D point cloud combination method, is introduced.

5.1 Robot Experiment Platform Setup

According to the descriptions in Section 4.1, the real experiment platform setup is kept consistent with the simulation platform. The whole system consists of an xArm6 six-axis robot arm, the end-effector with the cutting tool and Microsoft Azure Kinect RGB-D camera, a central control computer and a 3D-printed crab shell object. Figure 5.1 shows the robot experimental platform in the laboratory.

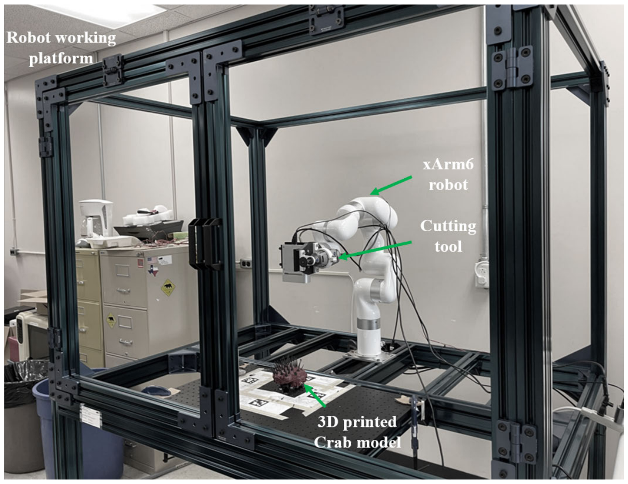


Figure 5.1: Robot experimental platform in laboratory

5.2 Robot Tool Frame Calibration

Section 3.3.1 describes the setting of the robot coordinate system. Some position errors may occur during the machining and assembly process of the end-effector in the real environment. In order to ensure that the planned trajectory can be executed accurately, the robot tool frame needs to be calibrated. In this thesis, the four-point calibration method is used [123]. Firstly, he robot calibration tool is fixed on the working platform, and the robot's tool center point(TCP) is attached to the center point of the disc-shaped cutting tool. Then we manually control the robot arm to make the TCP point reach the tip point of the calibration tool in four different poses. Using the four-point calibration method, the relationship between the robot tool frame $\mathcal{T}$ and the flange frame $\mathcal{F}$ can be determined, the result is shown in eq. (5.1), in which $\mathbf{R}_{\mathcal{T}}^{\mathcal{F}}$ and $\mathbf{p}_{\mathcal{T}}^{\mathcal{F}} = [DX, DY, DZ]^T$ are the rotational matrix and transnational vectors from the robot tool frame $\mathcal{T}$ to the flange frame $\mathcal{F}$. The forward kinematic method can calculate the relationship between the tool frame $\mathcal{T}$ and the global frame $\mathcal{B}$. Figure 5.2 shows the process of the four-point calibration method.

$$\mathbf{T}_{\mathcal{T}}^{\mathcal{F}} = \begin{bmatrix} \mathbf{R}_{\mathcal{T}}^{\mathcal{F}} & \mathbf{p}_{\mathcal{T}}^{\mathcal{F}} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & DX \\ 0 & 1 & 0 & DY \\ 0 & 0 & 1 & DZ \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

5.3 Camera Calibration

In this section, the 2D RGB and 3D depth cameras are calibrated to get the camera's intrinsic and distortion parameters to improve the accuracy of the captured image. After that, to obtain the pixel correspondence between the 2D RGB and 3D depth image, the transformation matrix for the correspondence between the 2D RGB and depth camera is calculated. Then, the coordinate relationship between the camera frame $\mathcal{C}$ and the robot flange frame $\mathcal{F}$ is determined by using the Hand-eye calibration method. The data captured by the camera can be transformed into the global coordinate frame for robot tool path planning.

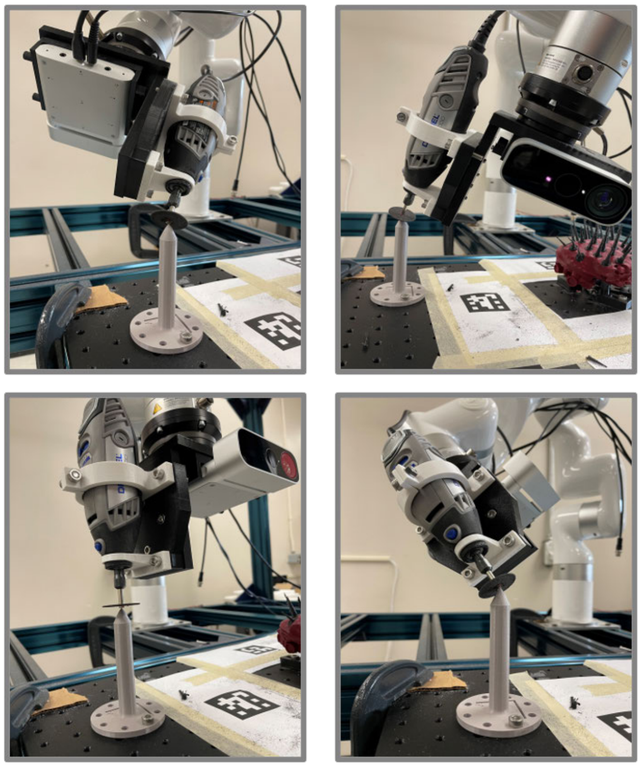


Figure 5.2: Robot tool frame calibration

5.3.1 2D RGB and 3D Depth Camera Calibration

Due to the unavoidable imperfection on the camera lens from manufacturing process, various forms of distortion are presented on the image. To remove the effect of distortion, the camera's distortion parameter needs to be calibrated.

This thesis uses Zhang's method [124] to calibrate the 2D RGB and 3D depth cameras. Firstly, the camera takes photos of the checkerboard from different angles, which are then imported into the C++ program for processing. This program applies the camera calibration function in the OpenCV API to identify the corner points of the checkerboard grid in the photos and calculate the camera intrinsic and distortion parameters. Figure 5.3 shows the captured image of the checkerboard. Equation((5.2)) and (5.3) shows the intrinsic **H** and distortion **D** parameters of the 2D RGB camera and 3D depth camera, which are written into the camera's firmware. The parameters can be automatically loaded by the camera during the image capturing to adjust the image accuracy itself.

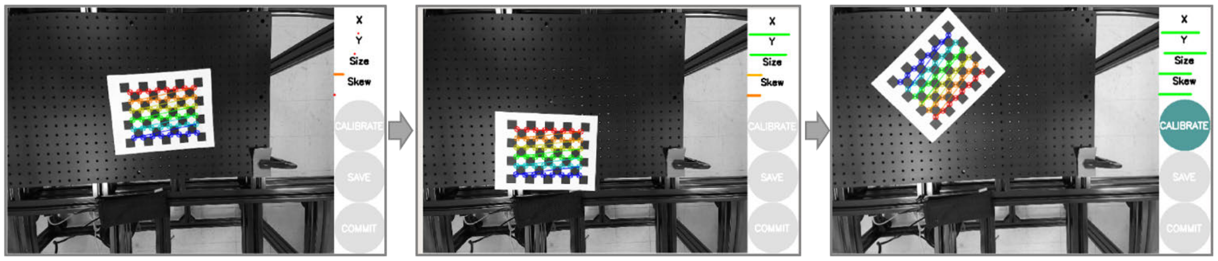


Figure 5.3: Camera calibration using checkerboard

$$\mathbf{H}_{rgb} = \begin{bmatrix} 612.654 & 0 & 612.709 \\ 0 & 635.7 & 368.03 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$
$$\mathbf{D}_{rgb} = [0.0121, 0.0625, 0.00313, -0.00524]$$

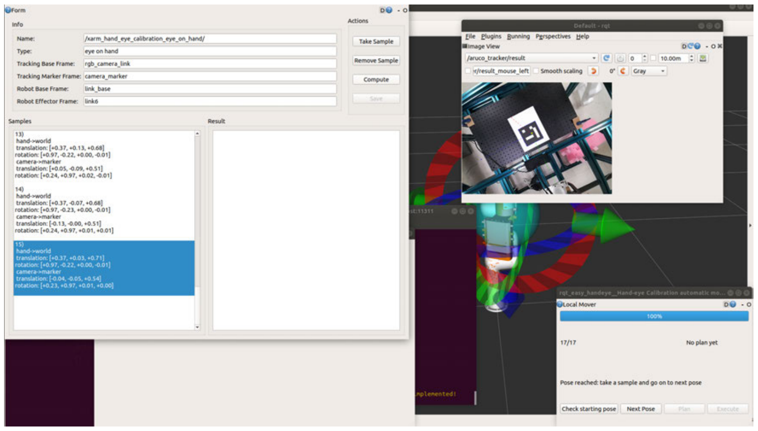
$$\mathbf{H}_{ir} = \begin{bmatrix} 376.238 & 0 & 224.136 \\ 0 & 314.330 & 264.011 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$
$$\mathbf{D}_{ir} = [0.0623, -0.120, 0.00210, -0.00410]$$

5.3.2 Hand-eye Calibration

In order for the robot to utilize the data collected by the camera, it is necessary to determine the transformation matrix between the camera frame $\mathcal{C}$ and the global frame $\mathcal{B}$.

This thesis uses the Hand-eye calibration method [125] to solve the transformation matrix between the camera frame $\mathcal{C}$ and the robot flange frame $\mathcal{F}$. An ArUco code is installed in a fixed position on the platform, and the camera can recognize the ArUco code to obtain the relationship between the ArUco code and the camera coordinate frame. The relationship between the robot flange frame $\mathcal{F}$ and global frame $\mathcal{B}$ is known. $\mathbf{X}$ is defined as the coordinate transformation matrix from the camera frame $\mathcal{C}$ to the robot flange frame $\mathcal{F}$. X can be solved efficiently by the Lie theory [126], which is encapsulated in the ROS package.

The hand-eye calibration for the Micorsoft Azure Kinect camera is shown in Fig. 5.4, the calibration result being shown in eq. (5.4).



Name:	/xarm_hand_eye_calibration_eye_on_hand/
Type:	eye on hand
Tracking Base Frame:	rgb_camera_link
Tracking Marker Frame:	camera_marker
Robot Base Frame:	link_base
Robot Effector Frame:	link6

```
13)
 hand->world
 translation: [+0.37, +0.13, +0.68]
rotation: [+0.97, -0.22, +0.00, -0.01]
 camera->marker
 translation: [+0.05, -0.09, +0.51]
rotation: [+0.24, +0.97, +0.02, -0.01]

14)
 hand->world
 translation: [+0.37, -0.07, +0.68]
rotation: [+0.97, -0.23, +0.00, -0.01]
 camera->marker
 translation: [-0.13, -0.00, +0.51]
rotation: [+0.24, +0.97, +0.01, +0.01]

15)
 hand->world
 translation: [+0.37, +0.03, +0.71]
rotation: [+0.97, -0.22, +0.00, -0.01]
 camera->marker
 translation: [-0.04, -0.05, +0.54]
rotation: [+0.23, +0.97, +0.01, +0.00]
```

```
/aruco_tracker/result   0   10.00m
r/result_mouse_left   0°
```

```
17/17                No plan yet

Pose reached: take a sample and go on to next pose
```

```
mplemented!
```

Figure 5.4: Robot hand-eye calibration

$$\mathbf{X} = \begin{bmatrix} 0.0007963 & -1.0000 & -0.0007963 & -0.02495 \\ 0 & 0.0007963 & -1 & -0.08686 \\ 1 & 0.0007963 & 0 & 0.07227 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.4}$$

5.4 Robot Spine Removal Experiment

•Robot spine removal experiment method

As mentioned in Section 4, the 2D image and 3D point cloud data are acquired by the Microsoft Azure Kinect camera. Using the 2D image and 3D point cloud combination method, the coordinates of the spines are identified, and the robot tool path is generated and smoothed. The robot's pose on the robot tool path is also calculated. In this section, the ROS control system is used to load the generated robot tool path and control the robot to execute this tool path to perform a spine removal experiment on a real 3D-printed crab shell. The correctness and validity of the robot spine removal method in this thesis are verified by analyzing the robot tool path execution and the quality of the crab model.

•Experimental result and analysis

Figure 5.5 shows the generated robot tool path using the 2D image and 3D point cloud combination method. Figure 5.6 shows the comparison between the robot tool path execution in the real experiment and the simulation environment. The results imply that the spine removal tool path is successfully planned and executed. The operating speed of the cutting tool is set to 35,000 rpm. The robot executes the spine removal tool path without shacking or collision, and all the spines on the crab shell can be cut off, which proves that the tool path is well adapted to the crab surface. The result demonstrates the accuracy of the robot tool path generation method. Figure 5.7 compares the crab shell before and after the spine removal process. The spine lengths before cutting are from 15 mm to 25 mm, and after the cutting, the lengths of all spines decrease to around 5mm with a smooth cross-section. Therefore, the risk of workers getting injured from processing the crab shell can be mitigated by this method.

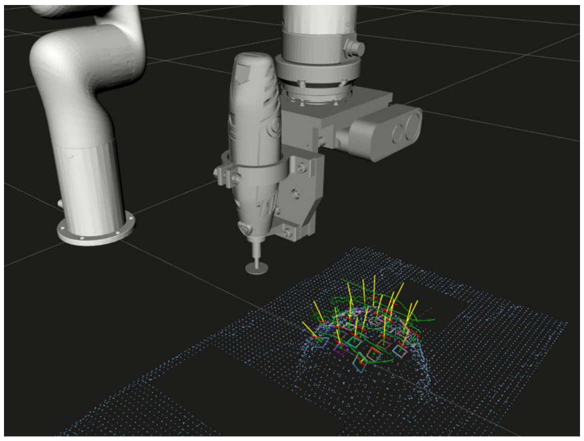


Figure 5.5: Generated robot spine removal tool path

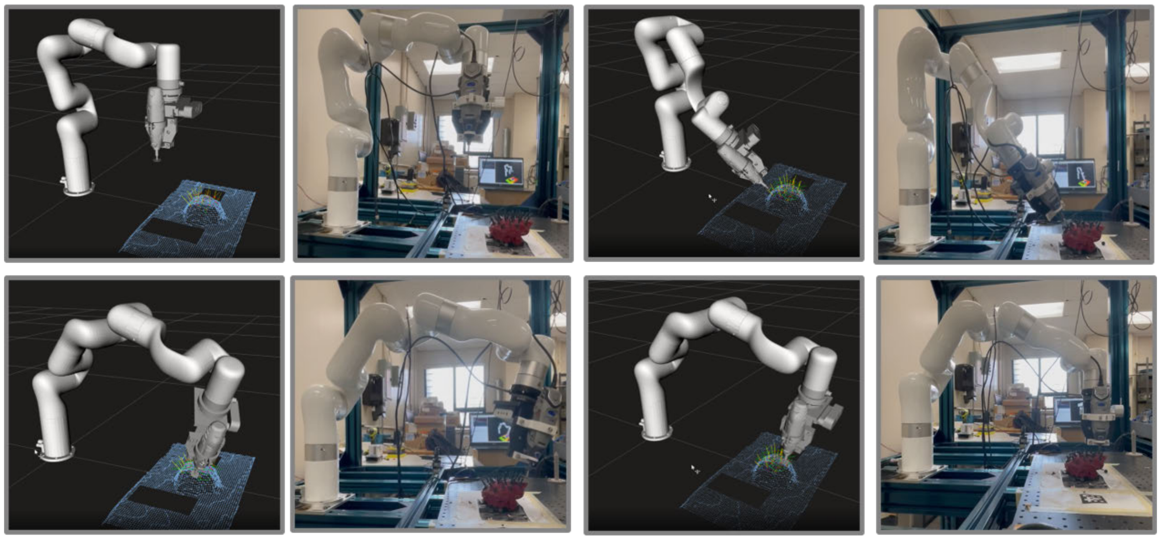


Figure 5.6: Comparison of the robot tool path in the simulation and real environment

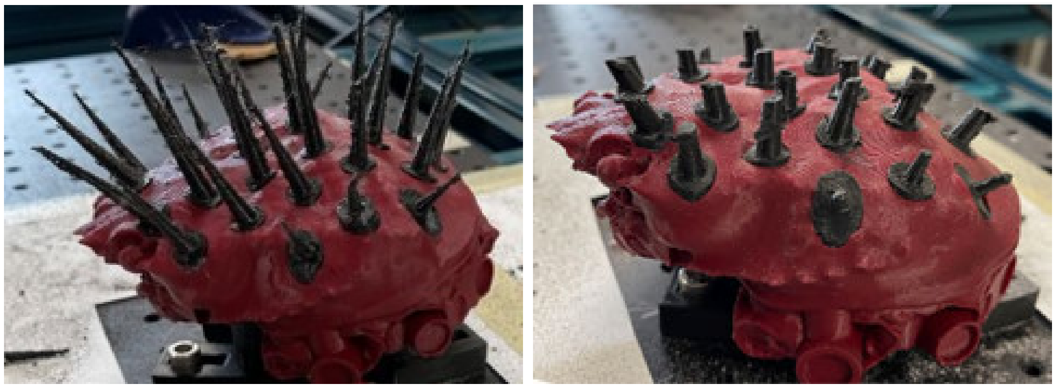


Figure 5.7: Crab model before and after spine removal process

5.5 Summary

This section introduces the experimental validation of the robot spine removal method. Firstly, the robot tool frame is calibrated to determine the relationship between the tool frame and the global frame. The camera intrinsic and distortion parameters are calibrated to minimize the data acquisition error. Furthermore, hand-eye calibration method is used to determine the relationship between the camera frame and the global frame. After completing the calibration of all parameters, the robot spine removal experiment is conducted, and the effectiveness and correctness of the thesis are verified.

Chapter 6

Conclusions and Future Work

6.1 Result Analysis

This thesis proposed a novel robot-based spine removal method to solve the difficulty in processing porcupine crabs, which is caused by the irregular spines on the crab body, and to assist in the commercialization of the crabs. Based on the 2D image and 3D point cloud data, this thesis proposed two novel methods to process the captured 2D and 3D data and plan the robot spine removal trajectory. The methodology has been simulated and verified in the ROS simulation environment and has been tested with the real robot arm in the real experiment environment. The main findings of this thesis are shown as follow:

(1) This thesis proposed a point cloud pre-processing method. The Microsoft Azure Kinect RGB-D camera obtains the crab body's actual 3D point cloud data. The point cloud data is processed by filtering, de-noising, surface normal calculation, registration and reconstruction. The complete 3D point cloud model of the porcupine

crab is well extracted and reconstructed from the redundant point cloud data.

(2) The first approach for robot spine removal tool path planning is a novel point cloud slicing and nearest points pair search method. This approach generates a robot tool path covering the whole area of the crab surface to remove all the spines. Although by using the first approach, all the spines can be removed, the accuracy and efficiency of this approach are lower than the second approach.

(3) The second approach for robot spine removal tool path planning is the 2D image and 3D point cloud combination method. Due to the parameters of the camera, the information of crab spines cannot be accurately captured. This approach firstly uses the 2D RGB image and 3D depth image to extract the features of spines in the 2D image. By using the ArUco code, the relationship between the 2D image and 3D point cloud data can be established. The spine features in the 2D image are then projected into the 3D point cloud model to calculate the spine root coordinate in the 3D point cloud model. The normal vector on the spine root coordinate need to be calculated, which will be used as the spine growth direction and to generate the robot tool path.

(4) After the robot tool path is planned, it is smoothed and optimized by using the cubic interpolation method. In order to verify the feasibility of the robot tool path, the simulation platform, as its real counterpart, is built inside the ROS simulation environment. The robot spine removal tool path generated from these two methods has been verified.

(5) A real experimental platform is built in the real laboratory environment. The spines on the 3D-printed crab model are successfully extracted and removed by the real robot system, which can verify the effectiveness of the proposed method. This

method can also be adopted to improve the automation level of the seafood production industry and many other industries.

6.2 Future Work

This thesis proposed and established a porcupine crab robot spine removal system and completed the experimental verification. There are still some improvements in space that can be further optimized.

(1) The spine removal parameters can be further optimized, including the optimization of cutting tool material, cutting speed, rotational tool speed and other parameters to enhance the quality of the product.

(2) The 2D and 3D vision-based spine extraction method has the potential to combine with the Machine Learning algorithm to improve the robustness of algorithm and recognition accuracy.

(3) The method presented in this thesis and the robot experiment platform are highly flexible. Future work can be focused on commercializing the platform and expanding the system application into various industries.

Bibliography

[1] Dassault System, “Solidworks,” May 2022. [Online]. Available: https://www.3ds.com/

[2] ROS.org, “Ros wiki,” May 2022. [Online]. Available: http://wiki.ros.org/Documentation/

[3] C. Bai, P. Dallasega, G. Orzes, and J. Sarkis, “Industry 4.0 technologies assessment: A sustainability perspective,” *International journal of production economics*, vol. 229, p. 107776, 2020.

[4] P. He, “Characteristics of bycatch of porcupine crabs, neolithodes grimaldii (milne-edwards and bouvier, 1894) from deepwater turbot gillnets in the northwest atlantic,” *Fisheries research*, vol. 74, pp. 35–43, Aug. 2005.

[5] E. Davidson and N. Hussey, “Movements of a potential fishery resource, porcupine crab (neolithodes grimaldii) in northern davis strait, eastern canadian arctic,” *Deep Sea Research Part I: Oceanographic Research Papers*, vol. 154, pp. 103–143, Dec. 2019.

[6] P.-Y. Chen, A. Y.-M. Lin, J. McKittrick, and M. A. Meyers, "Structure and mechanical properties of crab exoskeletons," *Acta biomaterialia*, vol. 4, no. 3, pp. 587–596, 2008.

[7] F. Dohrendorf, "Fish processing machine," U.S. Patent 4 084 294A, Apr. 18, 1978.

[8] Marel Inc, "Automatic salmon deheader ms 2720," May 2022. [Online]. Available: https://marel.com/en/

[9] RYCO Inc, "1230 crab processing container," May 2022. [Online]. Available: https://rycous.com/ryco-products/1230-crab-processing-container/

[10] BAADER Inc, "Baader 2801," May 2022. [Online]. Available: https://fish.baader.com/products/baader-2801/

[11] S. King and P. Hearn, "Sensor-guided automated method and system for processing crustaceans," U.S. Patent 10 264 799B2, Apr. 23, 2019.

[12] CBC(NL), "Can robots process crab?" May 2022. [Online]. Available: https://www.youtube.com/watch?v=RdmIe6qVG6I/

[13] Gonzalez Marketing LLC, "Bering fisheries ultimate king crab," May 2022. [Online]. Available: https://www.youtube.com/watch?v=vp_TG2bOSCM/

[14] Copper River Seafoods, "Copper river seafoods," May 2022. [Online]. Available: https://www.youtube.com/watch?v=vp_TG2bOSCM/

[15] I. F. Onstein, O. Semeniuta, and M. Bjerkeng, "Deburring using robot manipulators: a review," in *2020 3rd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS)*. IEEE, 2020, pp. 1–7.

[16] A. Beskopylny, A. Chukarin, P. Kurchenko, and A. Isaev, "Substantiation of systems for reducing noise and vibrations of grinding wheels in the processing of bodies of transport vehicles," *Transportation Research Procedia*, vol. 63, pp. 2529–2534, 2022.

[17] DMG Mori, "Cnc vertical multi-process grinding machine," May 2022. [Online]. Available: https://ca-en.dmgmori.com/products/machines/grinding/vertical-grinding/vertical-mate

[18] K. J. Brunete A, Gambao E, "Hard material small-batch industrial machining robot," *Robotics and Computer-Integrated Manufacturing*, vol. 54, pp. 185–199, 2018.

[19] M. Sabourin, F. Paquet, B. Hazel, J. Côté, and P. Mongenot, "Robotic approach to improve turbine surface finish," in *2010 1st International Conference on Applied Robotics for the Power Industry*. IEEE, 2010, pp. 1–6.

[20] J. N. Pires, J. Ramming, S. Rauch, and R. Araújo, "Force/torque sensing applied to industrial robotic deburring," *Sensor Review*, pp. 1–6, 2002.

[21] C. Ye and J. Borenstein, "Characterization of a 2d laser scanner for mobile robot obstacle negotiation," in *IEEE International Conference on Robotics and Automation*, vol. 3. IEEE, 2002, pp. 2512–2518.

[22] C.-H. Shih, Y.-C. Lo, H.-Y. Yang, and F.-L. Lian, "Key ingredients for improving process quality at high-level cyber-physical robot grinding systems," in *2020 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2020, pp. 1184–1189.

[23] Y. Dong, T. Ren, K. Hu, D. Wu, and K. Chen, "Contact force detection and control for robotic polishing based on joint torque sensors," *The International Journal of Advanced Manufacturing Technology*, vol. 107, no. 5, pp. 2745–2756, 2020.

[24] ABB Robotics, "Abb robotics stainless steel faucets (taps)," May 2022. [Online]. Available: https://www.youtube.com/watch?v=jXTJEEddOsY/

[25] Fanuc Robotics, "Robotic sanding, washing drying an aircraft fuselage with fanuc robot," May 2022. [Online]. Available: https://www.youtube.com/watch?v=abA9v8EOokI/

[26] C. Nguyen, J. Lee, and S. Yang, "Unified approach for force/position control in the car body sanding process," *The Korean Society for Fluid Power and Construction Equipment*, pp. 149–155, 2017.

[27] R. Bogue, "The growing use of robots by the aerospace industry," *Industrial Robot: An International Journal*, vol. 45, pp. 705 – 709, 2018.

[28] J. Liu, X. Huang, S. Fang, H. Chen, and N. Xi, "Industrial robot path planning for polishing applications," in *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2016, pp. 1764–1769.

[29] BARA, "Robot programming methods," May 2022. [Online]. Available: https://www.ppma.co.uk/bara/expert-advice

[30] J. L. L. T. A. J. E. P. Trygve Thomessen, Ole J. Elle and T. K. Lien, "Automatic programming of grinding robot," *MIC Journal Modeling, Identification and Control*, pp. 93–105, 1993.

[31] D. Guangyi, Z. Jinfu, and M. Kehua, "Study on polishing path generation method of mold curved surface based on mastercam," *Machinery*, vol. 45, pp. 46–48, Sep. 2007.

[32] M. Mohammad, V. Babriya, and T. Sobh, "Modeling a deburring process, using delmia v5," in *Technological Developments in Education and Automation*, Dordrecht, Netherlands, Dec. 2009, pp. 549–558.

[33] ABB Robotics, "Operating manual robotstudio," May 2022. [Online]. Available: https://library.e.abb.com/

[34] J. Muhovič, B. Bovcon, M. Kristan, J. Perš *et al.*, "Obstacle tracking for unmanned surface vessels using 3-d point cloud," *IEEE Journal of Oceanic Engineering*, vol. 45, pp. 786–798, May 2019.

[35] M. Dhanda and S. Pande, "Adaptive tool path planning strategy for freeform surface machining using point cloud," *Computer-Aided Design & Applications*, vol. 16, pp. 289–307, May 2019.

[36] A. Masood, R. Siddiqui, M. Pinto, H. Rehman, and M. A. Khan, "Tool path generation, for complex surface machining, using point cloud data," *Procedia CIRP*, vol. 26, pp. 397–402, Mar. 2015.

[37] X. Ren, G. Chen, Z. Wang, Z. Wang, and L. Sun, "Polishing path planning based on point cloud," in *Proceedings of the 2020 the 4th International Conference on Innovation in Artificial Intelligence*, Xiamen, China, May 2020, pp. 222–228.

[38] X. Zhen, J. C. Y. Seng, and N. Somani, "Adaptive automatic robot tool path generation based on point cloud projection algorithm," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Zaragoza, Spain, Sep. 2019, pp. 341–347.

[39] I. Robotics, "Industrial robot as defined by iso 8373," May 2022. [Online]. Available: http://www.ifr.org/industrial-robots/

[40] I. Akli and B. Bouzouia, "Time-dependant trajectory generation for tele-operated mobile manipulator," in *IEEE 3rd International Conference on Control, Engineering and Information Technology (CEIT2015), Algeria*, 2015.

[41] Z. Pandilov and V. Dukovski, "Comparison of the characteristics between serial and parallel robots." *Acta Technica Corviniensis-Bulletin of Engineering*, vol. 7, pp. 1–6, 2014.

[42] CAE, "Cae 7000xr series level d full flight simulator," May 2022. [Online]. Available: https://www.cae.com/

[43] ABB Robotics, “Irb 390 flexpacker,” May 2022. [Online]. Available: https://new.abb.com/products/robotics

[44] KUKA Robotics, “Kuka kr quantec robot,” May 2022. [Online]. Available: https://www.kuka.com/en-us/products

[45] UR Robotics, “Ur3: The world’s most flexible, light-weight table-top cobot to work alongside humans,” May 2022. [Online]. Available: https://www.youtube.com/watch?v=jsZvhDbnfRo

[46] ISO, “Iso 10218 1:2011 robots and robotic devices safety requirements for industrial robots,” May 2022. [Online]. Available: https://www.iso.org/standard/51330.html

[47] P. Waurzyniak, “Fast, lightweight robots help factories go faster,” *Manufacturing Engineering*, vol. 154, no. 3, pp. 55–64, 2015.

[48] I. F. of Robotics, “Demystifying collaborative industrial robots,” *Manufacturing Engineering*, vol. 5, pp. 65–74, 2018.

[49] ROS.org, “Ros robot operating system,” May 2022. [Online]. Available: https://www.ros.org/

[50] Automationware, “Programming with ros,” May 2022. [Online]. Available: https://automationware.it/ros-eng/?lang=en/

[51] ROS.org, “Ros core stacks,” May 2022. [Online]. Available: https://github.com/ros/

[52] ROS.org, “Ros docker,” May 2022. [Online]. Available: https://hub.docker.com/ros/

[53] ROS.org, “Rviz package,” May 2022. [Online]. Available: http://wiki.ros.org/rviz/

[54] Gazebo.org, “Gazebo simulation environment,” May 2022. [Online]. Available: https://gazebosim.org/home/

[55] Willow Garage, “Pr2 robot,” May 2022. [Online]. Available: https://robots.ieee.org/robots/pr2/

[56] ROS.org, “Moving robots into the future,” May 2022. [Online]. Available: https://moveit.ros.org/

[57] G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, and M. Beetz, “The robot household marathon experiment,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9382–9388.

[58] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. v. Stryk, “Comprehensive simulation of quadrotor uavs using ros and gazebo,” in *International conference on simulation, modeling, and programming for autonomous robots*. Springer, 2012, pp. 400–411.

[59] P. Katara, M. Khanna, H. Nagar, and A. Panaiyappan, “Open source simulator for unmanned underwater vehicles using ros and unity3d,” in *2019 IEEE Underwater Technology (UT)*. IEEE, 2019, pp. 1–7.

[60] ROBOTIS, "What is op3?" May 2022. [Online]. Available: https://emanual.robotis.com/docs/en/platform/op3/introduction/

[61] Xmachines, "Outdoor unmanned ground vehicle for robotics autonomy research," May 2022. [Online]. Available: https://www.xmachines.ai/research-platforms

[62] Coex, "Clover educational quadcopter kit," May 2022. [Online]. Available: https://coex.tech/clover

[63] Wikipedia, "Point cloud," May 2022. [Online]. Available: https://en.wikipedia.org/wiki/Point_cloud/

[64] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[65] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[66] Microsoft Inc, "Kinect for windows," May 2022. [Online]. Available: https://docs.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows

[67] Intel Inc, "Intel realsense technologys," May 2022. [Online]. Available: https://www.intel.com

[68] Asus Inc, "Xtion 2," May 2022. [Online]. Available: https://www.asus.com/ch-en/Networking-IoT-Servers/Smart-Home/Security-Camera/Xtion-2/

[69] L. Cherdo, “Handheld 3d scanners 2022: top 6 selection and buying guide,” May 2022. [Online]. Available: https://www.aniwaa.com/buyers-guide/3d-scanners/best-handheld-and-portable-3d-scanner/

[70] PCL.org, “Point cloud library (pcl),” May 2022. [Online]. Available: https://pointclouds.org/

[71] Q.-Y. Zhou, J. Park, and V. Koltun, “Open3D: A modern library for 3D data processing,” *arXiv:1801.09847*, 2018.

[72] O. Yilmaz, N. Gindy, and J. Gao, “A repair and overhaul methodology for aero-engine components,” *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 2, pp. 190–201, 2010.

[73] Institut maupertuis, “Automatic 3d grinding with an industrial robot,” May 2022. [Online]. Available: https://github.com/ros-industrial-consortium/bezier

[74] H. Wu, T. Zou, H. Burke, S. King, and B. Burke, “A novel approach for porcupine crab identification and processing based on point cloud segmentation,” in *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 1101–1108.

[75] G. Tzortzis and A. Likas, “The minmax k-means clustering algorithm,” *Pattern recognition*, vol. 47, no. 7, pp. 2505–2516, 2014.

[76] Dell Inc, “Xps desktop,” May 2022. [Online]. Available: https://www.dell.com/en-us/shop/cty/pdp/spd/xps-8940-desktop/

[77] Dremel Inc, “3000-1/24 variable-speed tool kit,” May 2022. [Online]. Available: https://www.dremel.com/ca/en/p/3000-1-24-f0133000ad/

[78] H. Senanayake, O. Akinsanmi, and M. B. Mu’azu, “An experimental autonomous path tracking mobile robot,” in *Advanced Materials Research*, vol. 62. Trans Tech Publ, 2009, pp. 181–186.

[79] B. He, X. L. He, L. Z. Han, J. T. Cao, M. Li, and Y. Z. Tian, “Working space analysis and simulation of modular service robot arm based on monte carlo method,” in *Applied Mechanics and Materials*, vol. 34, 2010, pp. 1104–1108.

[80] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo method.* John Wiley & Sons, 2016.

[81] P. I. Corke and O. Khatib, *Robotics, vision and control: fundamental algorithms in MATLAB.* Springer, 2011, vol. 73.

[82] Mathworks Inc, “Rand uniformly distributed random number,” May 2022. [Online]. Available: https://www.mathworks.com/help/matlab/ref/rand.html

[83] J. J. Craig, *Introduction to robotics: mechanics and control.* London, UK: Pearson, 2005.

[84] K. Szykiedans, W. Credo, and D. Osiński, “Selected mechanical properties of petg 3-d prints,” *Procedia Engineering*, vol. 177, pp. 455–461, 2017.

[85] Zortrax Inc, “Z petg materiel,” May 2022. [Online]. Available: https://store.zortrax.com/materials/m200-z-petg

[86] T. Foote, "tf: The transform library," in *IEEE International Conference on Technologies for Practical Robot Applications*, April 2013, pp. 1–6.

[87] Willow Garage, "Introducing openni," May 2022. [Online]. Available: http://wiki.ros.org/openni_camera

[88] D. Coleman, I. Sucan, S. Chitta, and N. Correll, "Reducing the barrier to entry of complex robotic software: a moveit! case study," *arXiv preprint arXiv:1404.3785*, 2014.

[89] ROS.org, "ros_control: A generic and simple control framework for ros," May 2022. [Online]. Available: http://wiki.ros.org/ros_control

[90] Orocos Kinematics and Dynamics, "Kdl wiki," May 2022. [Online]. Available: https://www.orocos.org/kdl.html

[91] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 928–935.

[92] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010. [Online]. Available: http://www.programmingvision.com/rosen_diankov_thesis.pdf

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

[94] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.

[95] Z. Ahmad and A. Guez, "On the solution to the inverse kinematic problem," in *Proceedings., IEEE International Conference on Robotics and Automation*. IEEE, 1990, pp. 1692–1697.

[96] J. C. A. Barata and M. S. Hussein, "The moore–penrose pseudoinverse: A tutorial review of the theory," *Brazilian Journal of Physics*, vol. 42, no. 1, pp. 146–165, 2012.

[97] P. Beeson and B. Ames, "Trac-ik: An open-source library for improved solving of generic inverse kinematics," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 928–935.

[98] S. Haddadin, A. De Luca, and A. Albu-Schäffer, "Robot collisions: A survey on detection, isolation, and identification," *IEEE Transactions on Robotics*, vol. 33, no. 6, pp. 1292–1312, 2017.

[99] T. Akenine-Möller, "Fast 3d triangle-box overlap testing," in *Acm siggraph 2005 courses*, 2005, pp. 8–20.

[100] M. Unser, A. Aldroubi, and M. Eden, "B-spline signal processing. i. theory," *IEEE transactions on signal processing*, vol. 41, no. 2, pp. 821–833, 1993.

[101] B. Dieber, R. White, S. Taurer, B. Breiling, G. Caiazza, H. Christensen, and A. Cortesi, "Penetration testing ros," in *Robot operating system (ROS)*. Springer, 2020, pp. 183–225.

[102] G. Zhang, J. Wang, F. Cao, Y. Li, and X. Chen, "3d curvature grinding path planning based on point cloud data," in *2016 12th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, Auckland, New Zealand, Oct. 2016, pp. 1–6.

[103] D. Garcia, "Robust smoothing of gridded data in one and higher dimensions with missing values," *Computational statistics & data analysis*, vol. 54, pp. 1167–1178, Apr. 2010.

[104] M. Pauly, M. Gross, and L. P. Kobbelt, "Efficient simplification of point-sampled surfaces," in *IEEE Visualization, 2002. VIS 2002.* IEEE, 2002, pp. 163–170.

[105] J. Sanchez, F. Denis, D. Coeurjolly, F. Dupont, L. Trassoudaine, and P. Checchin, "Robust normal vector estimation in 3d point clouds through iterative principal component analysis," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 163, pp. 18–35, May 2020.

[106] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE international symposium on mixed and augmented reality.* IEEE, 2011, pp. 127–136.

[107] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *Proceedings third international conference on 3-D digital imaging and modeling.* IEEE, 2001, pp. 145–152.

[108] F. Steinbrücker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense rgb-d images," in *2011 IEEE international conference on computer vision workshops (ICCV Workshops).* IEEE, 2011, pp. 719–722.

[109] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[110] D. Werner, A. Al-Hamadi, and P. Werner, "Truncated signed distance function: experiments on voxel size," in *International Conference Image Analysis and Recognition.* Springer, 2014, pp. 357–364.

[111] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *2011 IEEE international conference on robotics and automation.* IEEE, 2011, pp. 1–4.

[112] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics (ToG)*, vol. 36, no. 4, p. 1, 2017.

[113] T. Whelan, S. Leutenegger, R. Salas-Moreno, B. Glocker, and A. Davison, "Elasticfusion: Dense slam without a pose graph." Robotics: Science and Systems, 2015.

[114] R. Y. Tsai, R. K. Lenz *et al.*, “A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration,” *IEEE Transactions on robotics and automation*, vol. 5, pp. 348–358, Jun. 1989.

[115] S. Yuwen, G. Dongming, J. Zhenyuan, and L. Weijun, “B-spline surface reconstruction and direct slicing from point clouds,” *The International Journal of Advanced Manufacturing Technology*, vol. 27, pp. 918–924, May 2005.

[116] R. C. Gonzales and P. Wintz, *Digital image processing.* Addison-Wesley Longman Publishing Co., Inc., 1987.

[117] G. Deng and L. Cahill, “An adaptive gaussian filter for noise reduction and edge detection,” in *1993 IEEE conference record nuclear science symposium and medical imaging conference.* IEEE, 1993, pp. 1615–1619.

[118] K. J. Preacher and G. J. Leonardelli, “Calculation for the sobel test,” *Retrieved January*, vol. 20, p. 2009, 2001.

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

[120] A. P, F. L, M. G, and X. MĠ; *Applications of quaternions in robotics.* London, UK: Springer, 1998.

[121] L. Biagiotti and C. Melchiorri, *Trajectory planning for automatic machines and robots.* Springer Science & Business Media, 2008.

[122] K. Shoemake, “Animating rotation with quaternion curves,” in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, 1985, pp. 245–254.

[123] D. Whitney, C. Lozinski, and J. M. Rourke, “Industrial robot forward calibration method and results,” 1986.

[124] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[125] R. Y. Tsai, R. K. Lenz *et al.*, “A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration,” *IEEE Transactions on robotics and automation*, vol. 5, no. 3, pp. 345–358, 1989.

[126] F. C. Park and B. J. Martin, “Robot sensor calibration: solving ax= xb on the euclidean group,” *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 717–721, 1994.

Appendix A

Thesis code

Due to a large amount of code, the code of this thesis has been uploaded to GitHub for review.

(Link:https://github.com/WuRobotics/Novel-Point-Cloud-Based-Porcupine-Crab-Spine-Removal-Project).