Department of Mechanical Engineering

Implementation of ROS for a Quadruped Robot

Student Name: Chen Daoming

Student No: X00150770

Date of Submission: 3th May 2021

Supervisor: Paul Dillon

Declaration of Originality

I hereby certify, that this report, submitted as a part of the BEng (Hons) Mechanical Engineering, is entirely the work of the author and that any resources used for the completion of the work done are fully referenced within the text of this report.

Signature: Chen Daoming Date: 03/05/2021

Abstract

To make a mobile robot to conquer uneven surface, a legged robot is a wonderful approach. With the multiply axis legs, a robot can easily go upstairs, downstairs, and go through a rocky area. However, compared to a wheeled robot, the kinematics algorithms for a quadruped robot is much more complexed. This thesis mainly studies the implementation of ROS to control a quadruped robot moving in a virtual environment.

This thesis can be divided into four sections. Firstly, it introduces the current study of the quadruped robot and the platforms that are used to control the robot. In this project, the Boston Dynamic Spot robot and the ANYmal robot of ETH Zurich University are the main study objects. The platforms are MATLAB and ROS.

Secondly, this report offers some basic information on ROS, like the versions of ROS, the installation method for pc and raspberry pi and the working mode of ROS. By studying the basis, the results of this part are that the most suitable ROS version is ROS Melodic, and it is possible in theory to achieve letting a PC running as ROS master to wireless a mobile robot with a raspberry pi running ROS on it. However, this project doesn't achieve it in physics.

The third section is about the kinematics algorithm. This project is using 3 kinematics algorithm to achieve its function, namely, the forward kinematics algorithm, inverse kinematics algorithm and a trigonometry method. By applying these algorithms, the function of letting the robot walk forward can be achieved.

The last part of this project is developing a program to control the robot and simulate it in the Rviz simulator. To achieve that, a ROS package called champ(1), which is open-sourced on Github, can help with controlling the robot. Finally, the robot controlled by a keyboard walking straight can be simulated successfully.

Keywords: Quadruped Robot, Robot Kinematics, ROS

Acknowledgement

At the very end of my undergraduate life, I would like to express my sincere thanks to those who supported me and helped me pass through the difficulties.

First and foremost, I would like to thank my supervisor MR. Paul Dillion, who continuously give me suggestion and provide me with wonderful resources. At the very beginning of this project, I know nothing about how to manage a project on my own. Under the guidance of Mr Dillion, I made the schedule of this project, modified my objectives, started to know how to properly write a project report. Also, he helped me a lot with the learning process, by offering me resources and suggestion about learning ROS.

Secondly, I would like to thank GitHub and ROS. These two open-sourced platforms provide me with much information about how to make progress to this project. They make it possible to let someone like me with little knowledge background in Linux and ROS finish this robot project.

Last but not least, I would like to thank my friends who give me support when I failed in trouble. Especially, Chaonan Lin, who helped me a lot in this project by offering me ideas about how to solve a problem.

Table of Contents

CHAPTER 1 Introduction

1.1 Background

Some unique locations and circumstances in nature and modern society are inaccessible or too risky for humans, such as planetary surfaces, buildings, mines, catastrophe redemption and action against terrorism, etc. Nevertheless, it is still a significant long-term study to explore or implement work in those areas involving the improvements of science and technology. What those situations have in common are uneven surfaces for wheels or tracks to move. Compared with crawler robots, legged robots are more suitable for walking on rough terrains. The downside of legged robots is intricate mechanical design and control algorithms. Even walking on a flat surface requires coordination of complex calculations and mechanical structures. (2)

To improve this situation, robot developers need a platform where all kinds of technologies can meet perfectly. Therefore, robot developers need the Robot Operating System (ROS) as a forum for cooperation and technological advancement beyond the introduction and use of robots. ROS has different components for the spread and lowering of technology entry barriers(3). The Robot Operating System is a versatile framework for the writing of robot applications. It is a series of instruments, libraries, and conventions aimed at simplifying the task of developing complex and robust robot behavior acROSs a wide range of robotic platforms(4). In other words, ROS provides users with an operating system based on Linux to communicate with their robots, and by using ROS, users can simulate their robot's function. To install ROS, a Linux operating system like Ubuntu is required to be installed on a pc or a mobile system like Raspberry Pi. Then its multiple tools and packages can be used to build the human-computer interaction software and simulate the robot in the system.

1.2 Aims & Objectives

The aim of the project is to learn how to use MATLAB and ROS to create a specific algorithm controlling the quadruped robot's motion and use ROS to simulate its motion. After modelling and simulation, the robot will run in the real world.

The specific objectives of this project are listed as follows:

1. Learning algorithms

   A good many quadruped robots are existing in the world, like Spot(Made by Boston Dynamic) (Figure 1.1) ANYmal(Made by Swiss Federal Institute of Technology Zurich), and Laikago (Made by Unitree Robotics). They are good examples of quadruped robots and represents the highest level of human science and technology. It seems easy to let a quadruped robot walk. Nevertheless, there is a very complex programming algorithm behind every action.

Figure 1.1 Spot, made by Boston Dynamic (5)

2. Modelling the robot on SolidWorks

   The first step of building a robot is to complete a 3D model of the robot in software. SolidWorks is the only 3D model software supported by ROS. Thus, SolidWorks is used to build a robot model and export a model file with all the joint's parament to ROS.

3. Simulate the robot in ROS

   ROS is an open-source robot operating system with lots of packages to help users develop software controlling or simulating their robots. ROS is a complex system requiring a long-time investment in learning.

4. Controlling the real quadruped robot

   In this project, the mainboard, raspberry pi 4b, is used to run the ROS system for the mobile robot. To control the motors, an Arduino ATmega256 is involved. Raspberry pi is used as a communication and analysis device between a pc, which is used to send commands and run a simulation, and an Arduino.

1.3 Methodology

1. Design kinematics algorithm for the quadruped robot:
    - Obtain related knowledge by reading papers, website pages, and watching videos.
    - Design the kinematics algorithm for the robot and test it.

2. Build a 3D model by using SolidWorks:
    - Build a one-leg model on SolidWorks and motion test it
    - Develop the model of the entire robot
    - Export the model with motor parameters to ROS

3. Method of learning ROS:
    - Learning from ROS.org. There are many tutorials on ROS wiki
    - Reading books such as ROS Robot Programming (3)
    - Watching videos of ROS tutorial

4. Complete simulation by using ROS
    - Apply kinematics algorithm to program robot motion
    - Using SolidWorks to complete 3D model on ROS
    - Applying rviz to realize motion simulation

5. Assemble and control the robot in physics
    - 3D prints the robot and assemble them
    - Programming the robot
    - Testing the robot until it meets all the set functions

1.4 Project Plan

Semester 7

- Study kinematics algorithm of a quadruped robot
- Done most literature survey work
- Complete the 3D model on SolidWorks
- Simulation of the robot motion in Matlab
- Master the usage of ROS

- Create a primary model in ROS
- Complete report writing work including the above contents
- Complete the interim presentation

Semester 8

- Complete literature survey work
- Optimise the kinematics algorithm
- Create a 3D environment in Gazabo
- Complete motion simulation in Matlab
- Assemble the robot
- Testing the robot
- Discussion and Analysis
- Complete Report Writing
- Complete the final presentation

CHAPTER 2 Literature Survey

This literature survey is aimed to learn the current status of quadruped robots and in what method they are simulated and controlled. To control to robot's motion, what algorithm should be used? As well as two poplar robot program system, the Robot Operation System (ROS) and MATLAB.

2.1 Current Status of Quadruped Robots

Companies that currently dominate the quadruped robot field include Boston Dynamics, Google, Moog, Lynxmotion, ANYbotics, Foster-Miller, Arduino, Robotics, and Unitree. Among these companies' products, the most well-known dog-shaped robot is Spot, ANYmal, and Laikago (6). These three robots are similar in many ways but slightly different in size and function. The following will compare two of the robots, Spot, and ANYmal.

2.1.1 Spot(5)

Spot is a dog-shaped robot made by Boston Dynamics. Spot incorporates a stereo camera and depth cameras for environment mapping and route planning. It is waterproof and capable of working outdoors when the weather is not good. Depending on the tasks it performs, Spot can run for at most 2 hours

To provide input, Spot has a variety of sensors, including force and pressure proprioception sensors in its limbs that enable it to recognize the location of its limbs relative to itself, among other things. Spot, moreover, is fully electric, with no hydraulic actuators.

2.1.2 ANYmal(7)

ANYmal is manufactured by Robotics Systems Lab at ETH Zurich University. Different gait modes enable ANYmal to work on a wide range of both indoor and outdoor terrain. It is also ruggedized and sealed against both dust and water penetration. It can walk for 2 hours at its maximum walking speed, 1m/s. With the stereo cameras and LIDAR equipment, ANYmal can map its environment in 3D and autonomous localization and path planning.

To better help secure human-robot collaboration, this robot has been designed with torque controllability and compatible behaviour. All the joints of ANYmal are electric with no hydraulic actuators and can rotate 360°. It has an open-source modular software framework and can run on UBUNTU and ROS.

2.1.3 Comparing Results

Both Spot and ANYmal can move their bodies in roll, pitch, and yaw axis while keeping their articulated arm's end-effector stationary, which is a huge technical accomplishment for both robots. Over several terrain styles, they both have outstanding agility that involves manoeuvring up and down stairs. Moreover, the technology is very different inside of them. ANYmal, for example, relies heavily on torque sensors, while Spot uses more sensors for force and pressure.

2.2 Robot Operating System

Robot Operating System (ROS) is a software that builds for robots to integrate software and hardware. ROS is such a framework that provides us with rules and common forms to organize all other robotic platforms so that different software and people can work together on a large scale. Moreover, due to the highly open-source nature of ROS, developers can access lots of applications and libraries, which lower the barriers to developing robots(8).

2.2.1 ROS Version (3)

(i) ROS 1

ROS 1.0 was released on Jan 22, 2010. Every year there will be a newly released version of ROS. The latest version of ROS is ROS Noetic Ninjemys. It can only be installed on a Linux operating system, such as Ubuntu or Debian. The recommended system for ROS Noetic Ninjemys is Ubuntu 10.04.

ROS can offer some basic features to build a robot operating software for a robot. To make our robot more programable, ROS Toolbox should be involved in the project. Rviz plugin can help with visualizing the robot. Universal Robot description format (URDF) is made for importing 3D models to ROS. With all these tools, ROS is a super powerful platform to develop a robot project.

(ii) ROS 2

ROS 2.0 was released on Dec 08, 2017. Like ROS 1, each year, there will be a new version of ROS 2 released. The newest version of ROS 2 is ROS Foxy Fitzroy. It can be installed on Linux, Windows, and macOS. However, when installed on a non-Linux operating system, some ROS functions are limited. The recommended system for ROS Foxy Fitzroy is Ubuntu 10.04.

Like ROS 1, ROS 2 offers easy operating functions to build robot software. Besides, more tools can be plugged in ROS 2 and make it easier to use. The most significant difference between ROS and ROS 2 is that ROS 1 has a master, which is essentially the core. It is worth discussing whether querying the primary server is a "discovery" because it is a bit like a DNS server, which returns information in response to queries submitted to it by the client. However, ROS 2 does not have this at all. All nodes are nearly equal.

2.2.2 ROS working mode

(i) Packages (8)

The applications in ROS are grouped into packages. A package can contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or other things that are technically useful module. The aim of these packages is to offer this valuable functionality in an easy-to-use way so that the program can be reused quickly. All the packages are inside the workspace waiting to be called.

(ii) Nodes

To use the packages in ROS, the developer should first set up the packages in ROS master. Then, a publisher node makes a requirement of publishing a topic to the ROS master. A subscriber node will subscribe to the topic and receive the information about the publisher. Then the subscriber node will require a topic message from the publisher node. When the ROS message is successfully transferred from the publisher node to the subscriber node, the subscriber node will perform the function as the message said (Figure 2.1).

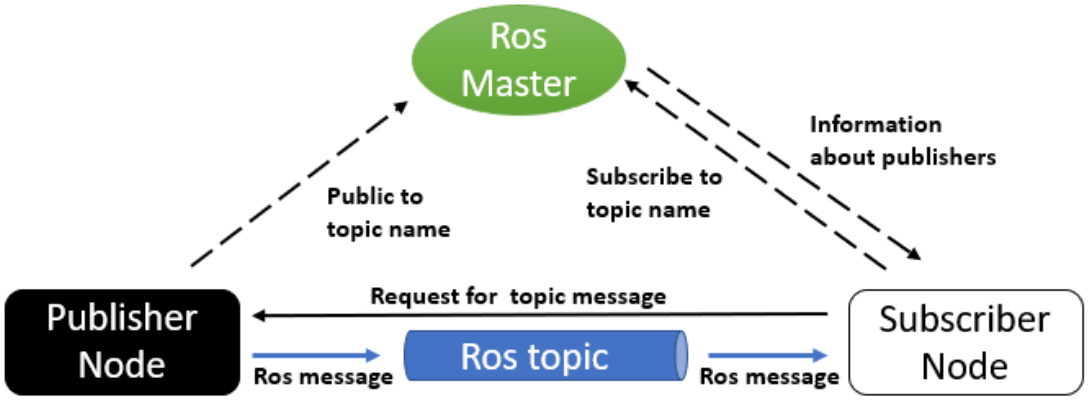


Figure 2.1 ROS node illustration diagram (9)

2.3 MATLAB Simscape(10)

Matlab Simscape is a physical block model environment based on Matlab Simulink. It enables users to build physical component models with physical connections. mechanical system, electrical system, fluid system is now supported in MATLAB Simscape (Figure 2.2). In the Simulink library, users can access many manpower saving tools like Deep Learning Toolbox, Control System Toolbox. With the help of these tools, users can rapidly complete their project in Matlab Simscape. Matlab offers users third party toolboxes such as ROS Toolbox(11) and Gazebo Toolbox(12). This would be helpful with a project that requires multiple software cooperation.

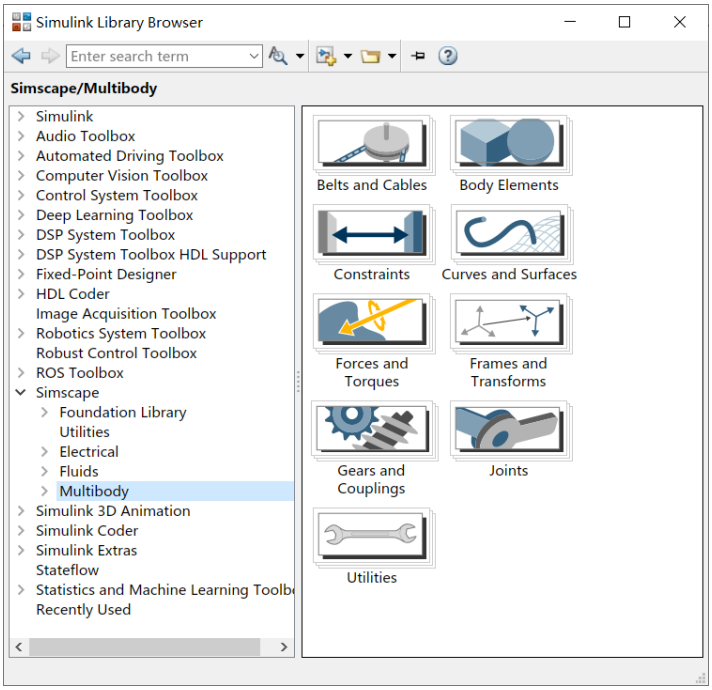


Figure 2.2 Useful tools in Matlab Simulink library

2.3.1 Matlab & ROS

Matlab Simulink provides developers a modular programming solution to build up libraries or packages in ROS. Compared to ROS's coding programming, Matlab Simulink reduced the difficulty of creating functional packages for robotic projects. Normally, a package can be built and verified by a visual method in Matlab Simulink. Hereafter, Matlab can automatically generate standalone ROS nodes in C++ from MATLAB and Simulink algorithms. Figure 2.3 displays how a robotic project is built

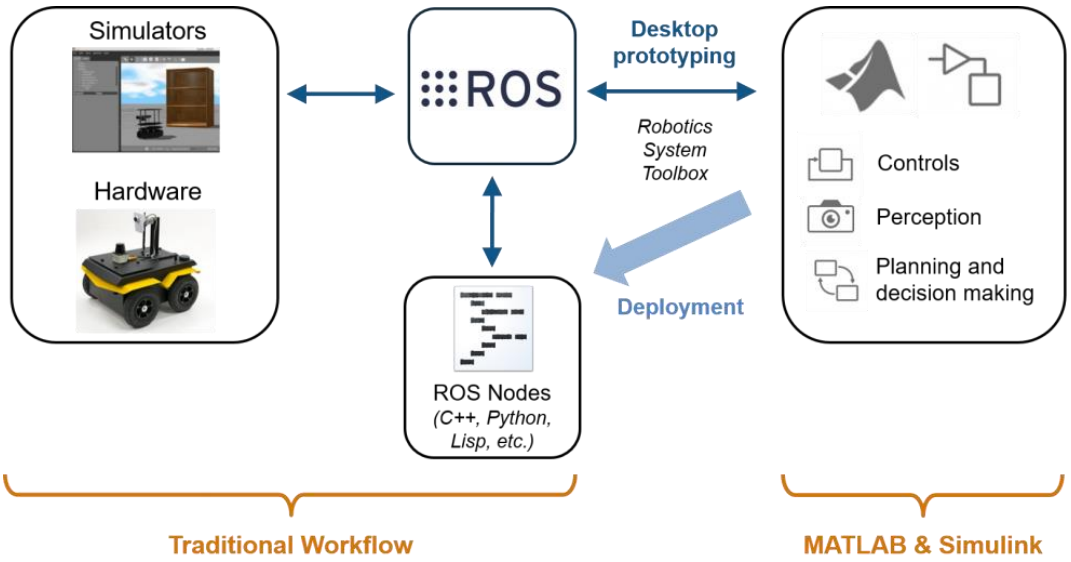


Figure 2.3 Matlab and ROS connection

2.4 Forward and Inverse kinematics(13)

Robot kinematics divided into forward kinematics and inverse kinematics. Forward kinematics is the solution to find the robot's end-effector when all the joint parameters are known. Inverse kinematics is the solution to find out every joint parameter of the robot when the end-effector position is known. Their relationship can be shown as follow.

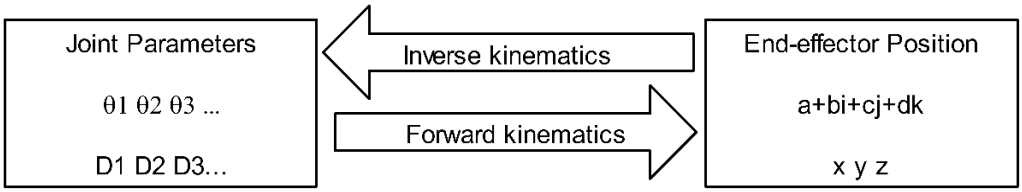


Figure 2.4 Graph of forward and inverse kinematics relationship

Denavit & Hartenberg (1955) put forward a solution to transform the two joints' coordinate with four parameters. This solution, known as the Denavit-Hartenberg (DH) matrix, is the most popular method to solve robot kinematics problems.

Forward kinematics problems, compares to inverse kinematics problems, are easier to solve. Figure 2.5 displays a simple example of a robot with multiple joints (Stanford Manipulator). To find the end-effector's position in the base coordinate, the DH matrix can be applied to this problem.

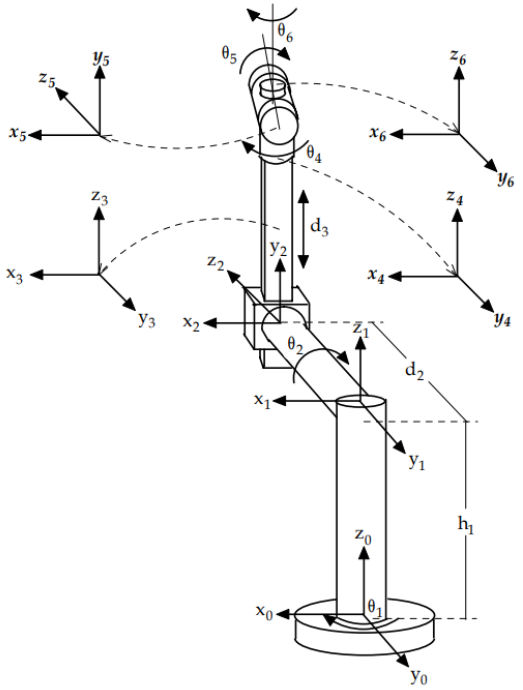


Figure 2.5(13) **Rigid body and coordinate frame assignment for the Stanford Manipulator**

(i) Step 1

To solve this kind of problem, we should first find out all parameters of the joints. Then, establish a table like Table 2.1. These parameters a_{i-1}, α_{i-1} , d_i and θ_i are the link length, link twist, link offset, and joint angle, respectively.

Table 2.1 DH parameters for the Stanford Manipulator

i	θ_i	α_{i-1}	a_{i-1}	d_i
1	θ_1	0	0	h_1
2	θ_2	90	0	d_2
3	θ_3	-90	0	d_3
4	θ_4	0	0	0

5	$\boldsymbol{\theta_5}$	90	0	0
6	$\boldsymbol{\theta_6}$	-90	0	0

(ii) Step 2

Apply the general transformation matrix ${}^{i-1}_{i}T$ to the robot links

Equation 1

$$
\begin{aligned}
{}^{i-1}_{i}T &= R_x(\alpha_{i-1})D_x(a_{i-1})R_z(\theta_i)Q_i(d_i) \\
&= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_{i-1} & -s\alpha_{i-1} & 0 \\ 0 & s\alpha_{i-1} & c\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & a_{i-1} \\ s\theta_i c\alpha_{i-1} & c\theta_i c\alpha_{i-1} & -sa_{i-1} & -sa_{i-1}d_i \\ s\theta_i s\alpha_{i-1} & c\theta_i s\alpha_{i-1} & ca_{i-1} & ca_{i-1}d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Where

- R_x and R_z present rotation
- D_x and Q_i denote translation
- $c\theta_i$ and $s\theta_i$ are shorthands for $cos\theta_i$ and $sin\theta_i$

(iii) Step 3

To find the end-effector's position in the base coordinate, all the transformation matrices should be multiplied together.

Equation 2

$$
{}^{base}_{end}T = {}^{0}_{1}T\ {}^{1}_{2}T \ldots {}^{n-1}_{n}T
$$

Equation 2 can also be written as

Equation 3

$$
{}^{base}_{end}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

In this particular case, we can derive that

$$
{}^{0}_{6}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Where

$$
r_{11} = -s\theta_6(c\theta_4 s\theta_1 + c\theta_1 c\theta_2 s\theta_4) - c\theta_6(c\theta_5(s\theta_1 s\theta_4 - c\theta_1 c\theta_2 c\theta_4) + c\theta_1 s\theta_2 s\theta_5)
$$

$$r_{12} = s\theta_6(c\theta_5(s\theta_1 s\theta_4 - c\theta_1 c\theta_2 c\theta_4) + c\theta_1 s\theta_2 s\theta_5) - c\theta_1 s\theta_2 s\theta_5) - c\theta_6(c\theta_4 s\theta_1 + c\theta_1 c\theta_2 s\theta_4)$$

$$r_{13} = s\theta_5(s\theta_1 s\theta_4 - c\theta_1 c\theta_2 c\theta_4) - c\theta_1 c\theta_5 s\theta_2$$

$$r_{21} = s\theta_6(c\theta_1 c\theta_4 - c\theta_2 s\theta_1 s\theta_4) + c\theta_6(c\theta_5(c\theta_1 s\theta_4 + c\theta_2 c\theta_4 s\theta_1) - s\theta_1 s\theta_2 s\theta_5)$$

$$r_{22} = c\theta_6(c\theta_1 c\theta_4 - c\theta_2 s\theta_1 s\theta_4) - s\theta_6(c\theta_5(c\theta_1 s\theta_4 + c\theta_2 c\theta_4 s\theta_1) - s\theta_1 s\theta_2 s\theta_5)$$

$$r_{23} = -\theta_5(c\theta_1 s\theta_4 + c\theta_2 c\theta_4 s\theta_1) - c\theta_5 s\theta_1 s\theta_2$$

$$r_{31} = -s\theta_6(c\theta_2 s\theta_5 + c\theta_4 c\theta_5 s\theta_4) - s\theta_2 s\theta_4 s\theta_6$$

$$r_{32} = -s\theta_6(c\theta_2 s\theta_5 + c\theta_5 c\theta_4 s\theta_2) - c\theta_6 s\theta_2 s\theta_4$$

$$r_{33} = c\theta_2 c\theta_5 - c\theta_4 s\theta_2 s\theta_5$$

$$p_x = d_2 s\theta_1 - d_3 c\theta_1 s\theta_2$$

$$p_y = -d_2 c\theta_1 - d_3 s\theta_1 s\theta_2$$

$$p_z = h_1 + d_3 c\theta_2$$

2.4.2 Inverse Kinematics

Inverse kinematics is the backward process of forwards kinematics. As it is illustrated above that the forward kinematics is to find the end effect position from the base. Inverse kinematics is to find the translation and rotation of each motor caused by the end effect motion. The formula of a 6-joint inverse kinematics is shown as follow:

Equation 4

$$ {}^0_6T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^0_1T(q_1)\ {}^1_2T(q_2){}^2_3T(q_3){}^3_4T(q_4){}^4_5T(q_5){}^5_6T(q_6) $$

Equation 5

$$ [{}^0_1T(q_1)]^{-1}{}^0_6T = [{}^0_1T(q_1)]^{-1}{}^0_1T(q_1)\ {}^1_2T(q_2){}^2_3T(q_3){}^3_4T(q_4){}^4_5T(q_5){}^5_6T(q_6) $$

where, $[{}^0_1T(q_1)]^{-1}{}^0_1T(q_1) = 1$, therefore the equation can be written as:

Equation 6

$$ [{}^0_1T(q_1)]^{-1}{}^0_6T = {}^1_2T(q_2){}^2_3T(q_3){}^3_4T(q_4){}^4_5T(q_5){}^5_6T(q_6) $$

By using the similar method as it is shown in Equation 6, we can obtain the following equations:

Equation 7

$$ [{}^0_1T(q_1)\ {}^1_2T(q_2)]^{-1}{}^0_6T = {}^2_3T(q_3){}^3_4T(q_4){}^4_5T(q_5){}^5_6T(q_6) $$

Equation 8

$$ [{}^0_1T(q_1)\ {}^1_2T(q_2){}^2_3T(q_3)]^{-1}{}^0_6T = {}^3_4T(q_4){}^4_5T(q_5){}^5_6T(q_6) $$

Equation 9

$$[{}_1^0T(q_1)\ {}_2^1T(q_2){}_3^2T(q_3){}_4^3T(q_4)]^{-1}{}_6^0T = {}_5^4T(q_5){}_6^5T(q_6)$$

Equation 10

$$[{}_1^0T(q_1)\ {}_2^1T(q_2){}_3^2T(q_3){}_4^3T(q_4){}_5^4T(q_5)]^{-1}{}_6^0T = {}_6^5T(q_6)$$

By solving the equation from the end effect to the base one joint by one joint according to the equation above, we can obtain the parameters for each joint.

CHAPTER 3 ROS Configuration

3.1 ROS Installation

Linux operating system is the recommended system to have ROS installed. Other systems like Windows or macOS runs ROS with limits. The most recommended Linux system for both PC and raspberry pi is Ubuntu.

3.1.1 Install ROS on personal computers

To install ROS on a personal computer requires the computer to run on a Linux system, such as Ubuntu. Linux is less user-friendly than other operating systems like Windows and macOS, therefore, it's better to run a Linux system as an optional system on a personal computer. (14)To do this, the Ubuntu system can be download on a USB drive and make it a bootable drive. When the driver is plugged into the computer, users can choose to boot the system either boot on the Ubuntu system or the original operating system. The second way to run the Ubuntu system as a secondary system on a personal computer is to use a virtual machine to run the Ubuntu system on your original system. However, this method requires a high-performance computer, because the PC is operating two systems at the same time.

When the Ubuntu system is available on a PC, check the Ubuntu version. Find a suitable ROS version from the ROS installation website. After that, there will be no problem installing ROS on a PC following the installation tutorial.

3.1.2 Install ROS on Raspberry Pis

To install ROS on a raspberry pi 4, the most important thing is to find the right operating system for pi first. From Raspberry Pi's official website users can download an imager (Figure 3.1) which can help to write and burn different kinds of Linux systems to an SD card (15). The recommended system is Ubuntu 20.04 because it is a long-term support system, and it is recommended by ROS Foxy Fitzroy and ROS Noetic Ninjemys. A newer version like Ubuntu 20.10 is much more user-friendly because it

provides pi users with a desktop. Nevertheless, ROS has not released an application version that can run on the latest version of Ubuntu. Therefore, the best choice is to write and burn Ubuntu 20.04 Sever ARM64 on an SD card with more than 16Gb memory.

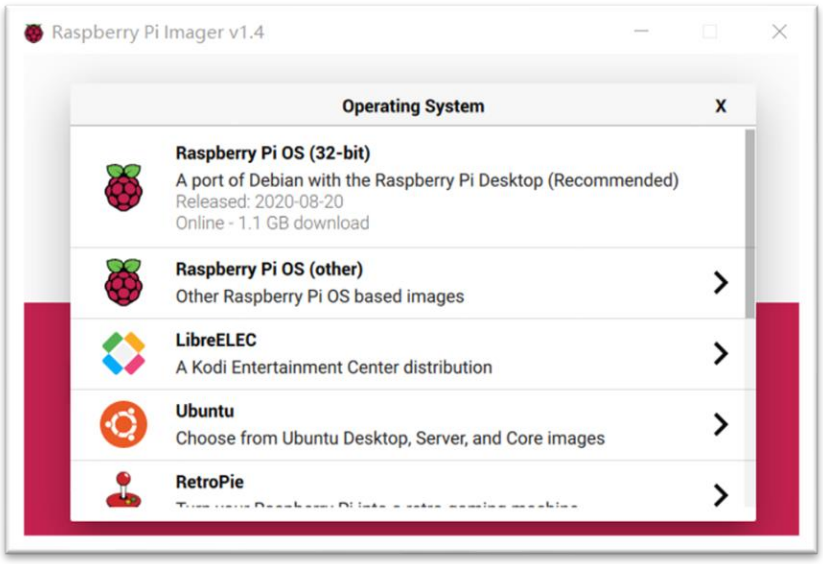


Figure 3.1 Raspberry Pi Imager interface

Before booting, the pi, the Pi requires a monitor to display and a keyboard to give commands. Power up the Pi, it will boot into a sever (Figure 3.2), which looks like a Dais system. Then, it requires the user to login. The original username for the system is ubuntu, and the password is also ubuntu. After the system confirmed login, the user is asked to change the password. The Ubuntu Server system works like a terminal, the user should use Dais code to communicate with the computer.

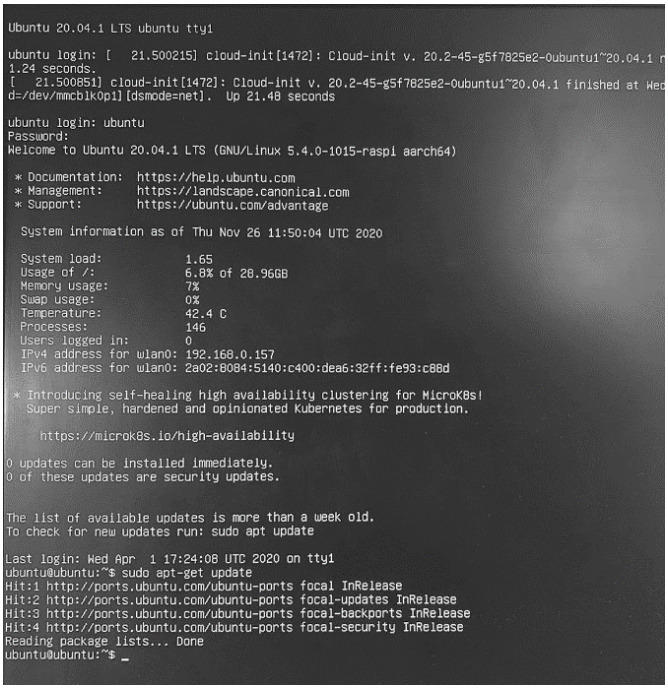


Figure 3.2 picture of Ubuntu 20.04 Server interface

The huge flaw of this system is that it cannot automatically connect to wifi. The user either uses a WLAN cable to connect the pi to the internet or add their local wifi information to the source file to let the pi accessible to the network (16).

To connect the pi to wifi, we should first check the name of the device:

```
Pi a
```

Eth0 corresponds to the wired network, and wlan0 corresponds to the wireless network. Then modify the configuration file:

```
sudo nano /etc/netplan/50-cloud-init.yaml
```

Open up the file and modify the file to the following style:

```
network:
    ethernets:
        eth0:
            optional: true
            dhcp4: true
    version: 2 #version 2 is a default value

    #add Wi-Fi setup
    wifis:
        wlan0:
```

```
            optional: true
            dhcp4: true
            access-points:
                "SSID": #SSID
                    password: "password" #password
```

Save changes and quit. Then generate and apply the network plan:

```
sudo netplan generate
sudo netplan apply
```

Reboot the computer:

```
sudo reboot
```

Two ways to check if the Wi-Fi is connected:

```
sudo apt-get update
```

if all the sources can be hinted at and applications updated normally, then it succeeded. The other way is to check the table with a row of wlan0 during the restart process, if there is a word “true”, the connection is successful.

After the Pi is connected to Wi-Fi, we can download a desktop for the system:

```
sudo apt-get install ubuntu-gnome-desktop
```

Then reboot the computer:

```
sudo reboot
```

Till now, all the Preparations are done for installing ROS 2 on Pi. What needs to be pointed out is that simply follow the tutorial on the ROS.org website will cause an error when installing ROS 2 on a pi. The following steps will guide the user to install ROS2(17). For ROS Noetic Ninjemys, following the installation tutorial works fine.

Set encoding:

```
1.  sudo locale-gen en_US en_US.UTF-8
2.  sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
3.  export LANG=en_US.UTF-8
```

Set the software source:

```
1.  sudo apt update && sudo apt install curl gnupg2 lsb-release
```

```
2.  curl -s https://raw.githubusercontent.com/ROS/ROSdistro/master/ROS.asc | sudo apt-key add –
```

Then,

```
1.  sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)] http://packages.ROS.org/ROS2/ubuntu $(lsb_release -cs) main" > /etc/apt/sources.list.d/ROS2-latest.list'
```

Install ROS2:

```
1.  sudo apt update
2.  sudo apt install ROS-foxy-desktop
```

Set environment variables:

```
1.  source /opt/ROS/foxy/setup.bash
```

Install auto-completion tool:

```
1.  sudo apt install python3-argcomplete
```

Run a few examples to test:

```
source /opt/ROS/foxy/setup.bash
ROS2 run demo_nodes_cpp talker
```

If the result is the same as, then ROS2 is successfully installed on the pi.

3.1.3 ROS version and package

When doing a robot project with ROS, it's really important to figure out the package version before using it. There are countless of open-sourced projects on GitHub, however, most projects can only run in one version of ROS. Be careful when trying to use the existing packages on GitHub to speed up the project. Because when the dependants of different ROS versions are different. For example, ROS Noetic, which is released in 2020, is using Python3 as its dependant but ROS Melodic, which is released in 2018, is using Python. That will cause the incompatibility between versions.

In this project, the most useful packages are champ and champ-telop, these two packages offer a robot joint operation method and control system for the robot. They are depended on the Python package which means they can only be used in ROS Melodic.

3.2 Arduino Set Up

In is project, Arduino is used as the real robot program translator. Since the Raspberry Pi 4 is used as the ROS core to publish and subscribe the ROS topics, the computation

capacity of it is running at the highest edge. It is necessary to using another motor control board to drive the robot. Arduino is the best choice. To use Arduino control by a Raspberry Pi, we need the following steps the configure it.

3.2.1 Arduino IDE Installation

There are 3 ways to install Arduino IDE on the Ubuntu system. The first way to install is to use apt-get to install:

```
Sudo apt-get upgrade
Sudo apt-get update
Sudo apt-get install Arduino
```

Even though the apt-get has been updated, by using those codes, an older version of Arduino IDE will be installed on your system. The older version means a lack of some new features.

Another way to use the ubuntu command to install Arduino IDE is to use the snap command.

```
Sudo snap install Arduino
```

By using snap means your Arduino will be automated installed under the directory, snap. This may cause trouble when installing libraries in the future. However, this time the Arduino IDE version is the latest.

The last way to install Arduino IDE is to download a package from the Arduino website then extract the package to the place you like. Before downloading the package, check your system version, normally PC need to download Linux 64bits, and Raspberry Pi 4b+ need to download Linux ARM 64bits. After extracting, open the terminal under the folder where the Arduino is extracted. Using root mode to extend authority to install the package.

```
// to enter root mode code in:
Sudo su
// using the shell file to install Arduino IDE
Sudo ./install.sh
// open Arduino
Arduino
```

3.2.2 Serial Port set up

Some machines may occur the error of can't find the serial port even through the Arduino is properly connected to the machine. Using the following code to extend port authority.

```
Dmesg | tail
```

3.2.3 Install ROS_Serial Library

There are several ways to install the ROS_serial library. The most convenient way is to use the library manager. By searching ROSserial and install version 0.7.9. However, this function can only be used if the Arduino IDE is the latest version.

CHAPTER 4 3D Model and Display in Simulator

4.1 3D Model Display

Displays the 3D model of the quadruped robot. As shown in Figure 4.1Table 4.1, there are 8 servos of the robot. Each leg is controlled by 2 servos. One of them controls the rotation of the shoulder, the other controls the rotation of the elbow. All four legs are designed to be elbow forward like Spot (5).

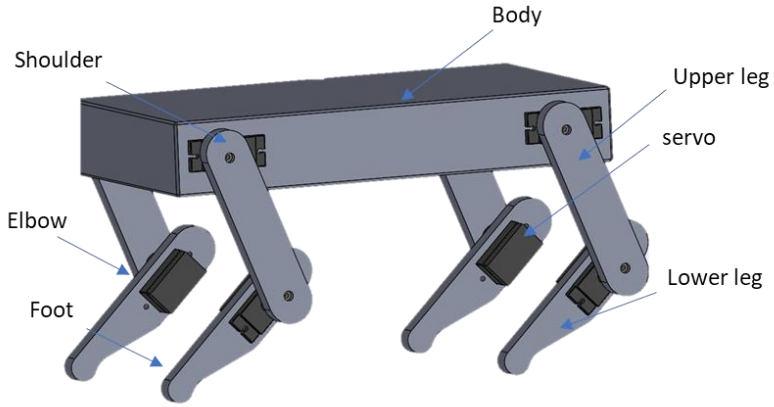


Figure 4.1 3D model of the robot

To design the length of the robot, the given length of the servo, SG90 (Figure 4.2). the size of the servo is 22mm*11.60mm*14.70mm of the bottom part with one 5mm diameter gear on the top and two 1mm diameter holes on each side. The body of the servo should be fixed on the robot body and the robot's lower legs. The gear of the servo should be fixed on the upper leg of the robot. The lower leg of the robot is designed to be 2.5 times longer than the servo's length. The length of the robot's upper leg is designed to be 3 times the servo's length. Therefore, the effective length of the upper leg is 61.60mm, and the lower leg is 54.54mm. To make the legs move without any collision with other legs, the body of the robot is designed to be a 187.74mm*109.50mm*30mm cuboid (Figure 4.3).

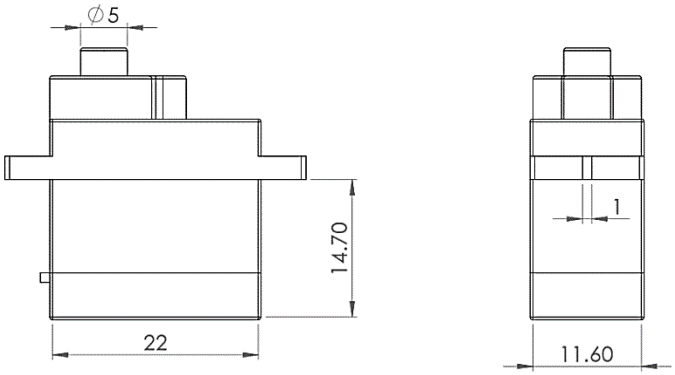


Figure 4.2 The model of SG90 (18)

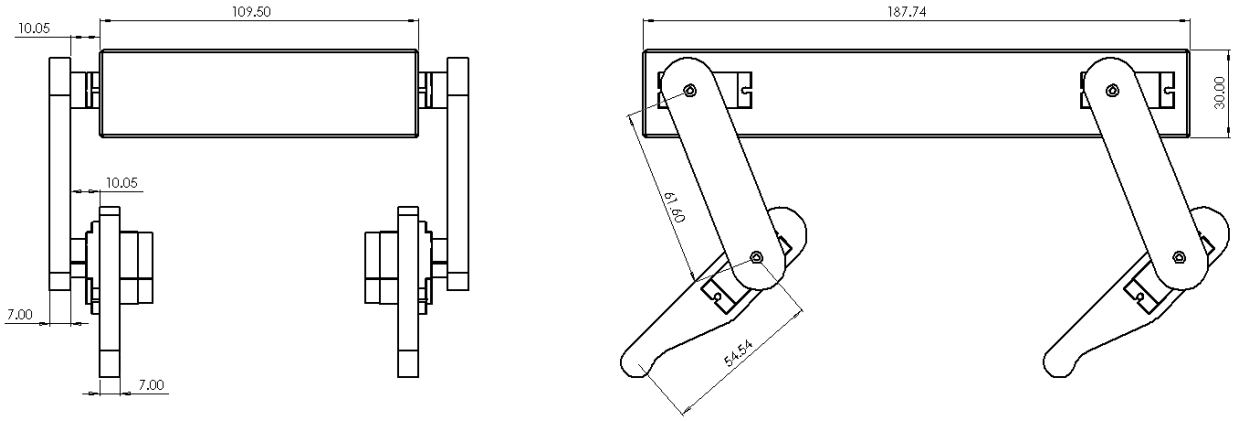


Figure 4.3 Drawing of the robot cat

4.2 Solidworks Model to URDF

4.2.1 URDF introduction

Unified Robot Description Format (URDF) is an XML specification used in academia and industry model multi-body systems, such as robot weapons manufacturing assembly lines and simulation of robot amusement parks. URDF is particularly popular with users of ROS, or robotic operating systems - a framework that provides standard support for URDF models. Like other types of xml files, URDF files contain various xml elements, such as <robot>, <link>, <joint>, and their coordinate in a 3d space nestled in a hierarchical structure called the XML tree. For example, <link> and <joint> elements are children of <robot> element.

```
<robot>
        <link>
                ...
        </link>
        <link>
                ...
        </link>
        <joint>
                ...
        </joint>
</robot>
```

4.2.2 URDF Generation Method

To convert a 3D Model to a URDF file, the only supportable 3D CAD software is SolidWorks. By using the sw_urdf_exporter plugin to achieve the function. Set origin, axis, and limits for the joints then click finish. The plugin software will automatically generate the .xml file. The .xml file can be loaded in MATLAB Simulink and ROS to describe the robot. Loading the .xml file into MATLAB Simulink and neat the model. Modify the joint parameters and set the solver to be ode23 to make the simulation can run a correct simulating result. Add the “special contact force” function to the model to make it stand on the ground. The robot operation simulation model is shown as follows.

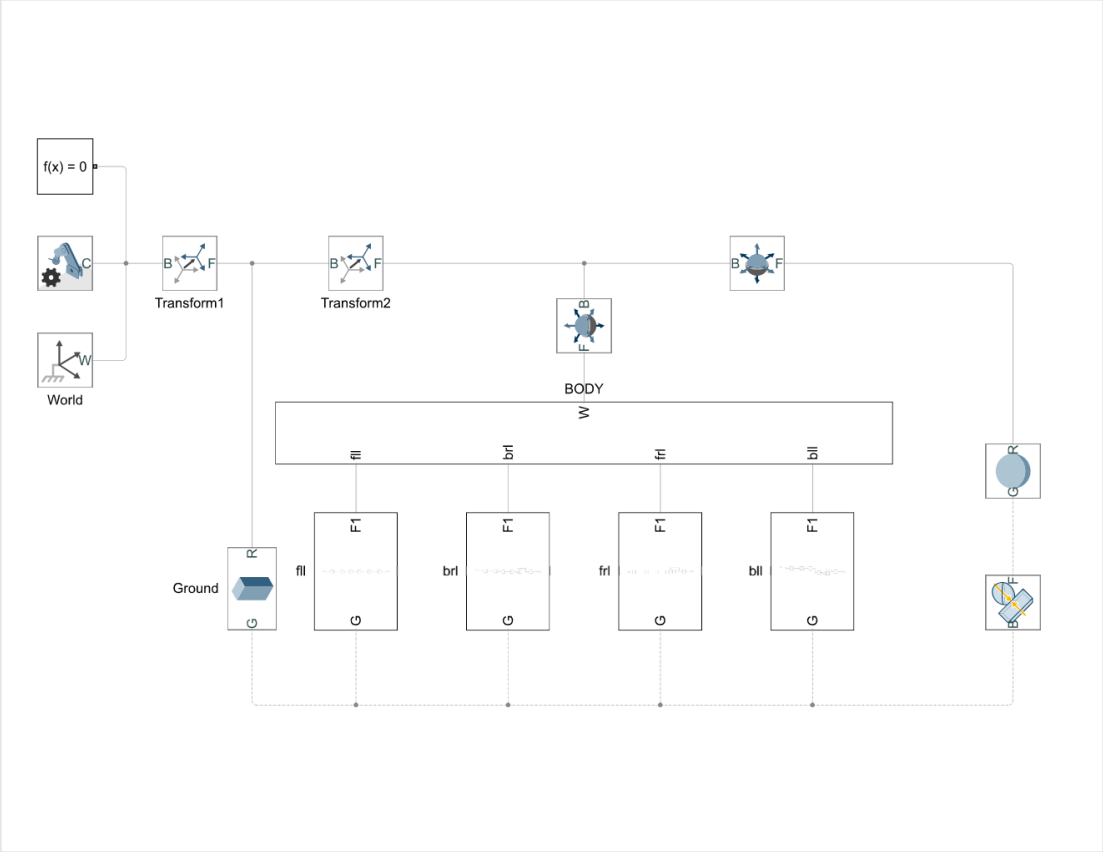


Figure 4.4 The model of URDF simulation in Matlab

Run the Simulink model then it will display the robot with physical contract with its surroundings.

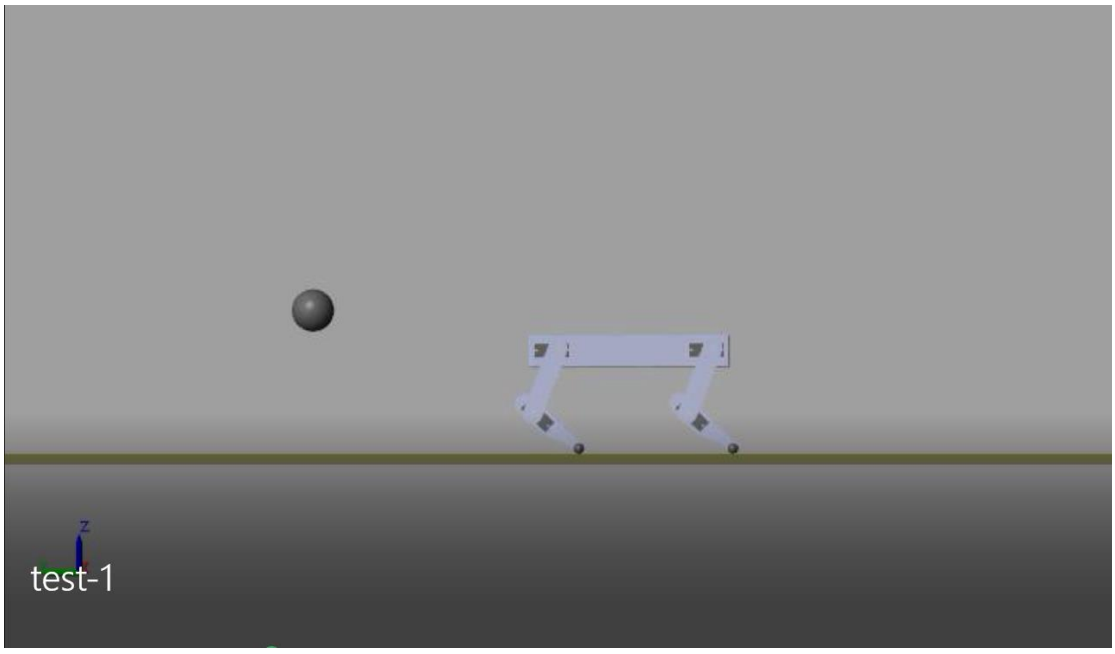


Figure 4.5 The simulation display of robot in Matlab

4.3 URDF in ROS

The sw_urdf_exporter can also create an urdf package that can be used in the ubuntu system, however, this package can't be directly opened in ROS, before some codes to be added to the package. The following steps show how to generate a URDF file for ROS.

In ROS, there is a build-in simulator, rviz. It provides a motion simulation environment but it doesn't have a real-world physical engine. That means, the kinematic control program can be tested in Rivz but can only provide a reference.

In order to display the robot in ROS, we can follow the steps below.

1. Create working space.

   Every package in ROS needs to be placed in a workspace so that the executable package path can be recorded by ROS. To achieve that, write the following code in the terminal.

```
sudo mkdir /home/urdf_test

# change the folder's characteristic

sudo chgrp username /home/urdf_test
sudo chown username /home/urdf_test

# create a new content

mkdir /home/urdf_test/catkin_ws/src
cd /home/urdf_test/catkin_ws/src
catkin_init_workspace

# Compile

cd /home/urdf_test/catkin_ws
catkin_make
```

2. Copy the package folder below the newly generated src folder

   The src folder is where all the packages placed. When calling a package by ROS command, the system will automatedly check if there is such a package in the src folder.

3. Modify two files

   Open the gazebo. launch file by vim and modify "robots" on line 13 to "urdf"

   Open the package.xml file and modify the contact email <maintaineremail="your email@xxxx.com"> to your email and add the author's name on the tenth line. It must be changed, otherwise, it will fail to compile.

4. Download an urdf.rviz file from GitHub

The URDF file can not be opened successfully unless there is a rviz driver. Luckily, it can be directly downloaded from Github.

5. Open with rviz with the following code.

```
$ROSlaunch display.launch
```

The following figure shows a robot model in URDF format downloaded from champ's GitHub repository(1). The reason for using this model is because champ is a mature package with a motion and control system which will be used in this project. The motion and control part of this package will be explained in the later chapters.

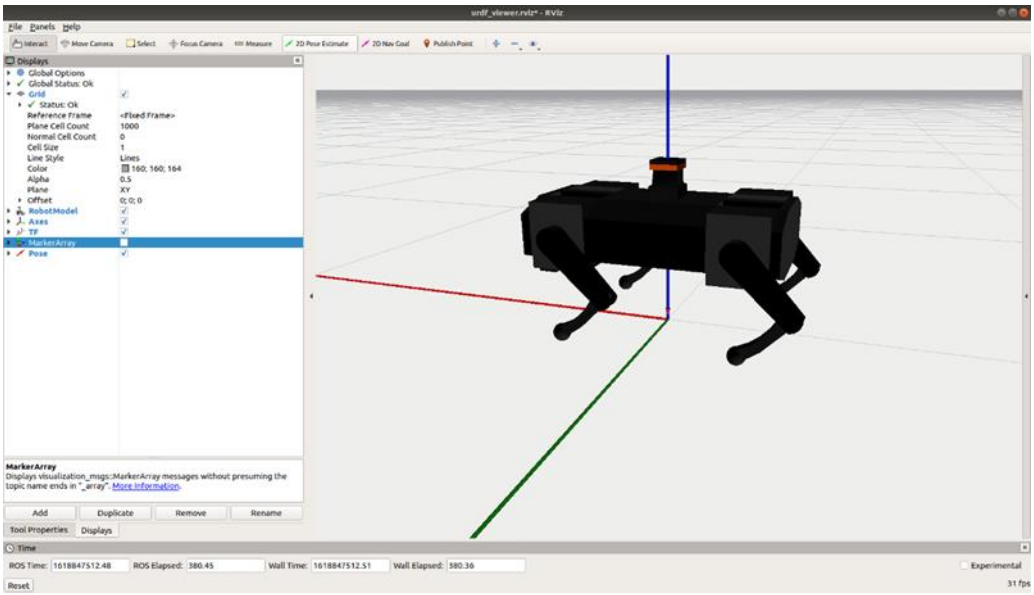


Figure 4.6 An URDF robot description in Rviz

CHAPTER 5 Kinematics of The Quadruped Robot

5.1 The kinematic solution to one leg(19)

To simplify the kinematic model, we consider only one leg at a time because the other three legs are moving the same way as this leg. After the motion of one leg is well planned, we then design the robot's motion to walk in harmony.

5.1.1 Two important initial conditions

To simplify the gait planning model, two initial conditions need to be proposed. First, the foot should always directly below the shoulder. Second, the angle between the upper leg and the lower leg needs to be 90°. When both the condition is satisfied, this position is called the home position.

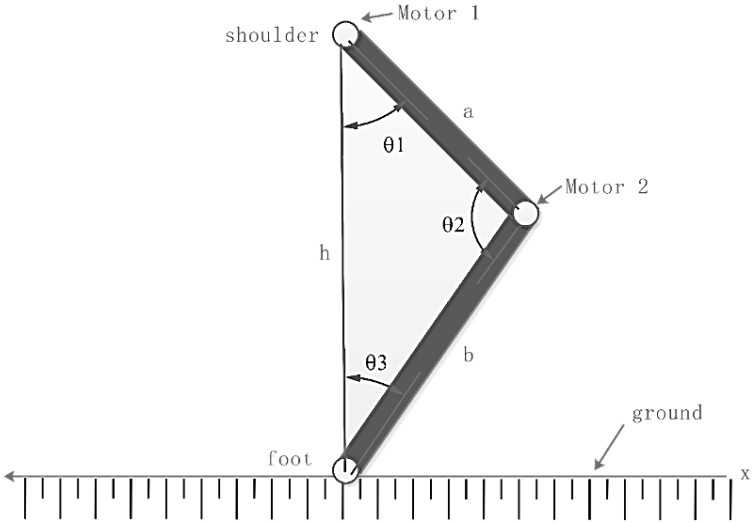


Figure 5.1 The home position schematics diagram

In Figure 5.1, a and b are the length of the upper leg and the lower leg, respectively. H is the distance between the shoulder and the foot. Line h is perpendicular to the ground. $\theta_1, \theta_2, \theta_3$ are the three angles of the triangle. Since a and b are known in the problem, the easiest to find h is to set $\theta_2 = 90°$. Then h can be calculated by the equation $h = \sqrt{a^2 + b^2}$.

The distance h is an input value in this model. When h is set at different values, the triangle's three angles can be easily found since it is a SSS triangle. Therefore, the motor's angular position can be set to meet the input h. Then the position shown in Figure 5.2 is named position 1.

Equation 11

$$\cos(C) = \frac{a^2 + b^2 - h^2}{2ab}$$

$$\cos(B) = \frac{a^2 + h^2 - b^2}{2ah}$$

$$\cos(A) = \frac{b^2 + h^2 - a^2}{2bh}$$

5.1.2 Gait planning

To let the robot move forward or backwards, we firstly set a step distance $+x$ or $-x$ for the leg, as Figure 5.2 shows. Then, we can easily find the distance between the shoulder and the new foot drop point d, $d = \sqrt{h^2 + x^2}$. The distance d will become our new side of the leg triangle, as Figure 5.3 displays. Afterwards, Equation 11 can be applied again to calculate the new value of $\theta_1, \theta_2, \theta_3$ of the new leg triangle and find the motor position of this posture. The position in Figure 5.3 is named position 2.

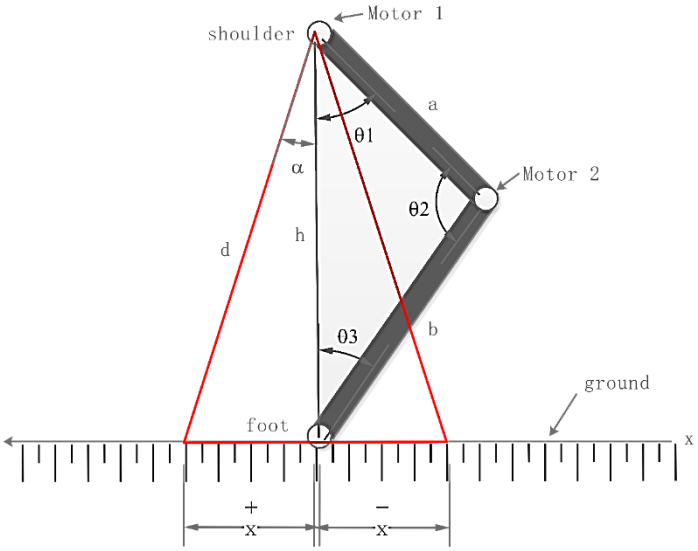


Figure 5.2 Gait planning schematics diagram, position 1

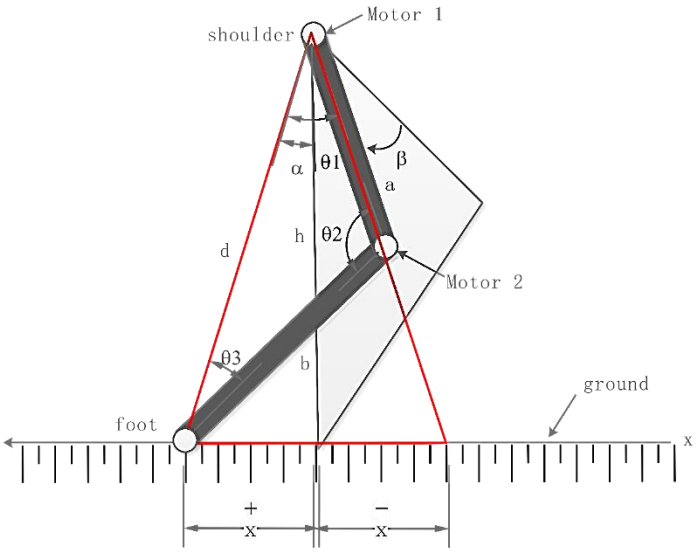


Figure 5.3 Gait planning schematics diagram, position 2

To make the leg move from position 1 to position 2, a middle position, position 3 is necessary. Figure 5.4 shows that from position 1 to position 3, the robot simply lifts its lower leg by 20° to make the foot off the ground.

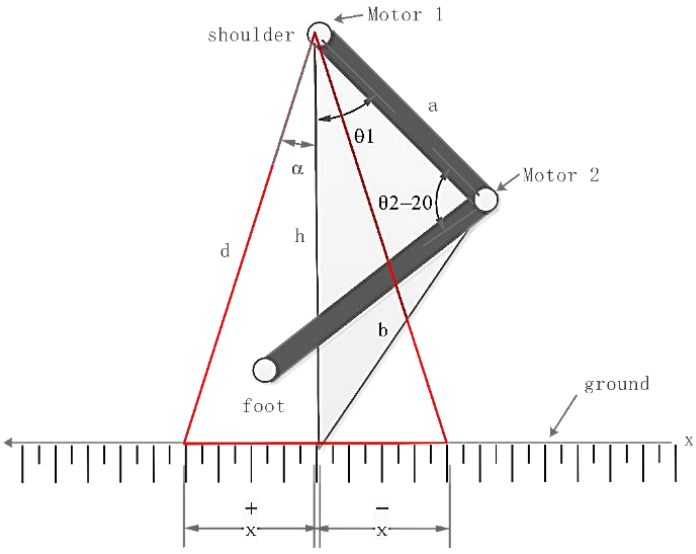


Figure 5.4 Gait planning schematics diagram, position 3

Table 5.1 shows the 2 motors' angle of the key position during the robot walking process when h is set at 110mm and x is set at 20mm.

Table 5.1 Motor angle of different positions when h = 110mm, x = 20mm

Position	The angle of Motor 1	The angle of Motor 2
Position 1	65°	55°
Position 2	68.2°	66.8°
Position 3	65°	75°

5.2 Pattern Generation

To move the robot forward or backwards. The leg movement order is required. The following shows how the robot moves in a specific order.

5.2.1 Motion generates in Solidworks

The motion generator in Solidworks is easy to use. Simply control the joints in mechanism function, manually set the joint actuation time and angle in order. The angle of each joint is calculated by the algorithm above.

By observing the cat's walking in slow motion. The motion order is set to be the back right leg (BRL) first, then the front left leg (FLL), then the back left leg (BLL), finally the front right leg (FRL). The back leg's motion is set to be ahead of the front leg by half a period. By applying the algorithm established above, when the step distance x is set at 20mm and the height h is set at 110mm. Then set the motion period for one leg at 2 seconds and the motion period for the robot should be 5s. Table 5.2, the joint angle value for the robot motion at different times, can be calculated.

Table 5.2 Joint angle value at different times when h = 110mm, x = 20mm

Joint Angle (degree) /Time (s)	0	0.3	0.6	0.9	1.2	1.5
FLL_Joint_1	65	65	65	68.2	68.2	68.2
FLL_Joint_2	55	55	75	66.8	66.8	66.8
FRL_Joint_1	65	65	65	65	65	68.2
FRL_Joint_2	55	55	55	55	75	66.8
BLL_Joint_1	65	65	65	65	68.2	68.2
BLL_Joint_2	55	55	55	75	66.8	66.8
BRL_Joint_1	65	65	68.2	68.2	68.2	68.2
BRL_Joint_2	55	75	66.8	66.8	66.8	66.8

(Note: F means front, B means back. The L in the middle means left, and the middle R means right. The last L means leg. Joint_1 and Joint_2 means shoulder and elbow respectively)

5.2.2 Motion generates in ROS(1)

To drive the URDF of the robot model in ROS, we need firstly let the simulation system acquire the robot's joints coordination in matrix format. The coordinate transformation has been addressed in 2.4 Forward and Inverse kinematics(13), we can use a 4x4 matrix to show the transformation of a joint. From CHAPTER 4, a robot description file has been generated with the geometry information of the robot. According to the geometry data from the URDF, a header file that contains the joint information acquired from the URDF is required. The following figure shows the code to achieve this.

```
namespace geometry
{
    class Point : public Matrix<3,1>
    {
        public:
            Point() { Fill(0); }
            Point(const Point &obj) : Matrix<3,1>() { (*this) = obj; }
            Point(const Matrix<3,1> &obj) { (*this) = obj; }
```

Figure 5.5 Geometry header file code of coordinate

The "obj" in the code represent the coordinate acquired from the URDF file. The coordinate is a 3x1 vector contains the parameter x, y, z. Then the parameters will be saved in the "*this" matrix to let it be operated by transformation operation function. Then According to ROS system's working mode, all the parameters need to be published and subscribed to achieve its function.

Besides the coordinate, the rotation method information needs to be appeared in the 3x3 matrix. The following figure shows the method to achieve that. Phi, Theta and Psi represent the angle in x, y, and z axis. Again, all the information needs to be published to a ROS topic. Besides of the codes in Figure 5.6, Euler Angles is here to correct the matrix in rotation process. When (2,0) of "*this" matrix is not 0 or it equal to -1, the matrix needs to be corrected as it is shown in Figure 5.7.

```
class Rotation : public Matrix<3,3>
{
    public:
        Rotation() { *this = Identity<3,3>(); }
        Rotation(const Rotation &obj) : Matrix<3,3>() { (*this) = obj; }
        Rotation(const Matrix<3,3> &obj) { (*this) = obj; }

        Rotation &FromEulerAngles(float psi, float theta, float phi)
        {
            (*this)(0,0) = cosf(phi) * cosf(theta);
            (*this)(1,0) = cosf(theta) * sinf(phi);
            (*this)(2,0) = -sinf(theta);

            (*this)(0,1) = cosf(phi) * sinf(psi) * sinf(theta) - cosf(psi) * sinf(phi);
            (*this)(1,1) = cosf(psi) * cosf(phi) + sinf(psi) * sinf(phi) * sinf(theta);
            (*this)(2,1) = cosf(theta) * sinf(psi);

            (*this)(0,2) = sinf(psi) * sinf(phi) + cosf(psi) * cosf(phi) * sinf(theta);
            (*this)(1,2) = cosf(psi) * sinf(phi) * sinf(theta) - cosf(phi) * sinf(psi);
            (*this)(2,2) = cosf(psi) * cosf(theta);

            return (*this);
        }
```

Figure 5.6 Geometry header file code of rotation

```
Matrix<3,2> ToEulerAngles()
{
    Matrix<3,2> ret;

    if((*this)(2,0) != 0)
    {
        ret(1,0) = -asinf((*this)(2,0));
        ret(1,1) = M_PI - ret(1,0);

        ret(0,0) = atan2f((*this)(2,1) / cosf(ret(1,0)),(*this)(2,2) / cosf(ret(1,0)));
        ret(0,1) = atan2f((*this)(2,1) / cosf(ret(1,1)),(*this)(2,2) / cosf(ret(1,1)));

        ret(2,0) = atan2f((*this)(1,0) / cosf(ret(1,0)),(*this)(0,0) / cosf(ret(1,0)));
        ret(2,1) = atan2f((*this)(1,0) / cosf(ret(1,1)),(*this)(0,0) / cosf(ret(1,1)));
    }
    else
    {
        ret(2,0) = ret(2,1) = 0;

        if((*this)(2,0) == -1)
        {
            ret(1,0) = ret(1,1) = M_PI_2;
            ret(0,0) = ret(0,1) = atan2f((*this)(0,1),(*this)(0,2));
        }
        else
        {
            ret(1,0) = ret(1,1) = -M_PI_2;
            ret(0,0) = ret(0,1) = atan2f(-(*this)(0,1),-(*this)(0,2));
        }
    }
```

Figure 5.7 Correction method for rotation of the Geometry.h file

When the coordinate is changing, the temporary matrix of the joint needs to be recorded. As it is addressed above that, Phi, Theta and Psi represent the rotation in x, y, z axis respectively. Take the x axis as an example. When the joint is rotating theta, the matrix will change as Figure 5.8 illustrated. For y and z axis, the change will be similar.

```
Rotation &RotateX(float phi)
{
    float tmp1, tmp2;

    tmp1 = (*this)(1,0) * cosf(phi) - (*this)(2,0) * sinf(phi);
    tmp2 = (*this)(2,0) * cosf(phi) + (*this)(1,0) * sinf(phi);
    (*this)(1,0) = tmp1;
    (*this)(2,0) = tmp2;

    tmp1 = (*this)(1,1) * cosf(phi) - (*this)(2,1) * sinf(phi);
    tmp2 = (*this)(2,1) * cosf(phi) + (*this)(1,1) * sinf(phi);
    (*this)(1,1) = tmp1;
    (*this)(2,1) = tmp2;

    tmp1 = (*this)(1,2) * cosf(phi) - (*this)(2,2) * sinf(phi);
    tmp2 = (*this)(2,2) * cosf(phi) + (*this)(1,2) * sinf(phi);
    (*this)(1,2) = tmp1;
    (*this)(2,2) = tmp2;

    return (*this);
}
```

Figure 5.8 Rotate in X axis of the Geometry.h file

Till now, the kinematics program of the robot has been generated. After all these calculations of the joint position, we can get a reference position for the robot's motion. Then, all those results need to be sent to the joint_states topic so that the controller part of this project can receive the results and get command to the robot under the kinematics rules.

CHAPTER 6 Robot Motion Controller Package Explain

6.1 Controller package(1)

Till now, the project's library has been complete. However, it can only provide the robot display with the joint position transformation method. The kinematics method is there, but it requires a controller to move the robot. PID control method is used in this package, and the parameters are set to be P = 180, D = 0.9, I = 20. Furthermore, the kinematics method program requires this to be linked with ROS, which is a bit complex and will be explained in the following text.

6.1.1 Controller executable

The executable file is the file that can be run by ROS and achieve the function. An executable file can be written in CPP or python. For this project, the executable file is written in CPP.

According to the kinematics library above, there are three things left to control the motion of the robot. The first one is the joint position capture executable. To make the control system a closed-loop, joint position capture is indispensable. It can always provide the accurate position of the joint in the simulation environment and compare the position with the calculated one to provide input parameters for the calculation and be the correct tool for the calculation. The second executable is the controller executable. It is used to be the gait model selection tool. However, in this project, there is only one gait being used, and that is forward with a constant step distance of 1 unit. The last executable is the message relay executable. Since this is quite a complex ROS package with a mass of messages send around. A stand-alone executable to make sure the messages are sending to the right node is essential. The following content will talk about each of these executables.

(i) Position capture executable

Firstly, this executable is subscribed to the ROStime topic, which can provide this simulation with an accurate reference time. Then subscribe to the joint_states topic, to receive messages of the calculated joint position. To capture the joint position in a simulation environment, it requires the odometry package (Odom) to capture the

position. Luckily, Odom is a built-in package in ROS melodic. Compare these 2 position coordinates, if they are the same, then publish the result directly. If not, there are a few steps to correct the joint position. Because of the geometry limitation of the robot leg, it not likely to encounter an error in relative position. In other words, the related coordinate between the foot and the joint is correct. The most possible mistake comes from the joint position in the world coordinate. As it is known that the joint position is calculated by the inverses kinematics method. Therefore, by using the forward kinematics, the joint position in the world coordinate can ensure to be correct. Another likely mistake in joint position is the foot is not placed on the ground. This can be easily detected by Odom. As long as the z-axis value of the foot point is not 0, the foot is not placed on the ground.

This problem can be fixed by replacing the current z with 0 and recalculate the inverse kinematics. Then publish the corrected joint position.

(ii) Controller executable

Still, the first step for building this controller executable is to subscribe to the ROStime package to provide reference time when simulation. This is where the controller system gets the input parameters. The parameters are shown in Figure 6.1. As we can see that the parameters aren't a number but a source. The reason for not directly input the number is because whenever the project changes, it requires a compilation, and that takes time. If we can let the program acquire a parameter from a configuration file, it will save time. For this project, the configuration file where the parameters are stored is a "Yet Another Markup Language" file, (YAML file). The YAML file is a relatively new configuration file format and it is easy to understand. Therefore, it is much more convenient to place parameters that require a frequent change in a YAML file. Figure 6.2 shows the context of the YAML file. Another function of this executable file is chosen gait. However, in this project, we only study the straight forward gait with constant step distance.

```
QuadrupedController::QuadrupedController(ros::NodeHandle *nh, ros::NodeHandle *pnh):
    body_controller_(base_),
    leg_controller_(base_, rosTimeToChampTime(ros::Time::now())),
    kinematics_(base_)
{
    std::string joint_control_topic = "joint_group_position_controller/command";
    std::string knee_orientation;
    double loop_rate = 200.0;

    nh->getParam("gait/pantograph_leg",          gait_config_.pantograph_leg);
    nh->getParam("gait/max_linear_velocity_x",   gait_config_.max_linear_velocity_x);
    nh->getParam("gait/max_linear_velocity_y",   gait_config_.max_linear_velocity_y);
    nh->getParam("gait/max_angular_velocity_z", gait_config_.max_angular_velocity_z);
    nh->getParam("gait/com_x_translation",       gait_config_.com_x_translation);
    nh->getParam("gait/swing_height",            gait_config_.swing_height);
    nh->getParam("gait/stance_depth",            gait_config_.stance_depth);
    nh->getParam("gait/stance_duration",         gait_config_.stance_duration);
    nh->getParam("gait/nominal_height",          gait_config_.nominal_height);
    nh->getParam("gait/knee_orientation",        knee_orientation);
    pnh->getParam("publish_foot_contacts",       publish_foot_contacts_);
    pnh->getParam("publish_joint_states",        publish_joint_states_);
    pnh->getParam("publish_joint_control",       publish_joint_control_);
    pnh->getParam("gazebo",                      in_gazebo_);
    pnh->getParam("joint_controller_topic",      joint_control_topic);
    pnh->getParam("loop_rate",                   loop_rate);
```

Figure 6.1 Controller input parameters

```
1   knee_orientation : ">>"
2   pantograph_leg : false
3   odom_scaler: 0.9
4   max_linear_velocity_x : 0.5
5   max_linear_velocity_y : 0.25
6   max_angular_velocity_z : 1.0
7   com_x_translation: 0.0
8   swing_height : 0.04
9   stance_depth : 0.00
10  stance_duration : 0.25
11  nominal_height : 0.20
```

Figure 6.2 The parameters in the YAML file

(iii) Message relay executable

Message relay is a relatively simple executable. The main purpose is to determine what information should the message have and in what format. Where is the information come from and where should it be sent to? How large is the size of the message?

6.1.2 Controller node

For each executable, it requires a node to let messages from executables be sent around. It enables the executable to publish and receive messages. In this project, 3 main nodes correspond to the 3 main executables.

The node is much simpler than the executable script, the following figure shows the format of the quadruped controller node. Other nodes are almost the same as this one. The only difference is the included library and the node name.

```
#include "ros/ros.h"
#include "quadruped_controller.h"

int main(int argc, char** argv )
{
    ros::init(argc, argv, "quadruped_controller_node");

    ros::NodeHandle nh("");
    ros::NodeHandle nh_private("~");

    QuadrupedController champ(&nh, &nh_private);

    ros::spin();
    return 0;
}
```

Figure 6.3 Format of a node file

6.2 Tele operation program

With all those things that have been built, there is one thing left before the robot can be controlled as wish in the virtual environment. That is an operating system. To write this program, it requires all the message from the controller part and makes the command from the teleoperation part as the input to the controller.

To achieve that, the teleoperation program should first subscribe to the nodes that are mentioned in the former content. Then set the specific key position on the keyboard with a defined function. The operation method is set as follows:

```
Reading from the keyboard and Publishing to Twist!
---------------------------
Moving around:
```

```
u    i    o
j    k    l
m    ,    .
For Holonomic mode (strafing), hold down the shift key:
---------------------------
U    I    O
J    K    L
M    <    >
t : up (+z)
b : down (-z)
anything else : stop
q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
CTRL-C to quit
```

It is not hard to write those operation methods since the control system for the robot has been established. The only thing that needs to be done to achieve this program is to define the key to move the robot.

6.3 Launch file

6.3.1 Introduction

The Launch file is provided by ROS, which can run files of multiple nodes at the same time. Launch files are written in a special XML format and are widely used in ROS packages. The format of a launch file is shown as follows:

```
<launch>
    <arg name="argument's name"                    default="different
according to the arg" />


    <node name="node name" pkg="package name" type="type of the package"
output="can be ignored" />
        <remap from="new name for a package" to=" new name for a package "
/>
    </node>
</launch>
```

The arg is short for argument, which contains the information that is required by the node. Then node name and the package point out the nodes and packages that are involved in the launch file. There are normally more than one nodes and packages listed in the launch file. When the launch file is finished, by typing the following codes in the terminal, then the program starts to run.

```
ROSlaunch package_name launch_file_name
```

6.3.2 Launch files in this project

Many launch files existed in the champ package(1). However, for this project, only two launch files are enough. The first one is the model description with the controller launch file. The nodes that are involved are the robot's urdf node and the nodes for the controller.

The second one is the teleoperation launch file. This launch file only involved the teleoperation node.

6.3.3 The result of starting the launch file

By entering the following codes in the terminal, the robot will be shown in rviz with an operation panel. The results are shown in Figure 6.4.

```
ROSlaunch champ_config bringup.launch rviz:=true
ROSlaunch champ_teleop teleop.launch
```

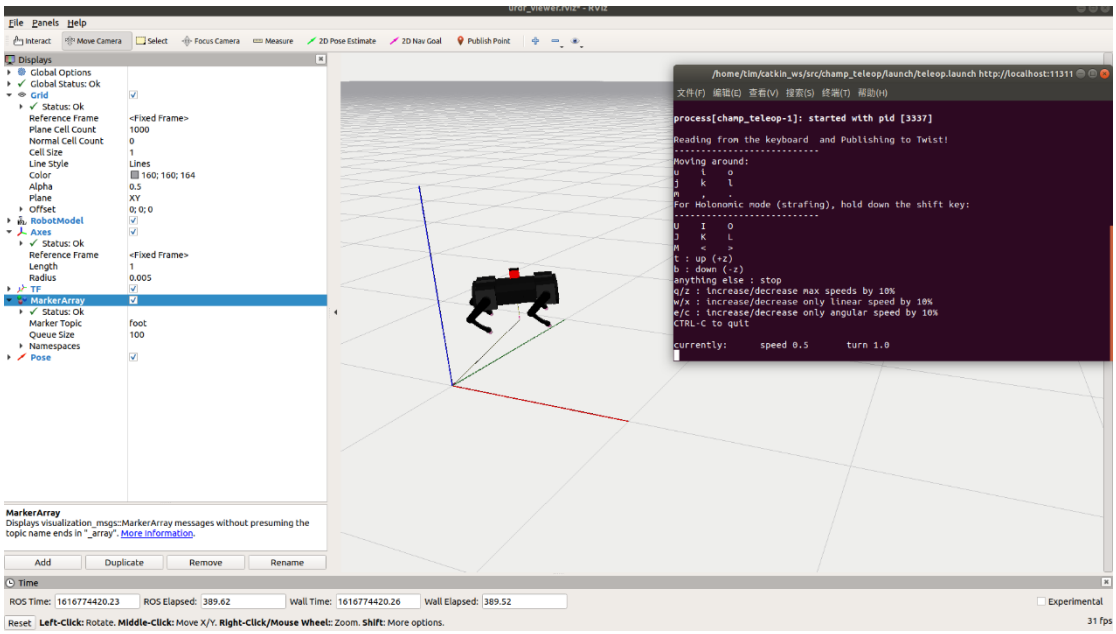


Figure 6.4 Robot in rviz and controlled by a keyboard

6.4 Summary

To conclude what is going on when the two launch files are running, there is a ROS package called RQT to help demonstrate the situation. RQT is a built-in tool of ROS that can provide users with GUI. The following content will illustrate the result of two rqt tools.

6.4.1 RQT graph

RQT graph is one of the most useful RQT tools. The rqt graph can display the subscribe relationship between nodes with the delivered message on the arrow line. The following codes show how to run the rqt graph

```
ROSrun rqt_graph rqt_graph
```

When running the rqt graph package after the two launch files started, the nodes that are active and the relationships between them can be seen in Figure 6.5.

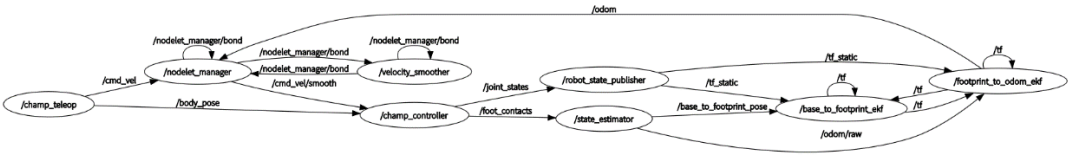


Figure 6.5 ROS rqt diagram of demonstrating the nodes and their relationships

6.4.2 RQT_tf tree graph

RQT tf tree provides a GUI plugin for ROS to visualize the TF frame tree. TF is a tool to help track the many coordinates of the robot in a virtual environment. Run the following codes, it will show the graph of the TF frame tree of the current robot position. It can also tell the developer about a list of coordinates that can be tracked of the robot and where the coordinate is based on. In addition, the TF tree can be refreshed as the robot moves.

```
ROSrun rqt_tf_tree rqt_tf_tree
```

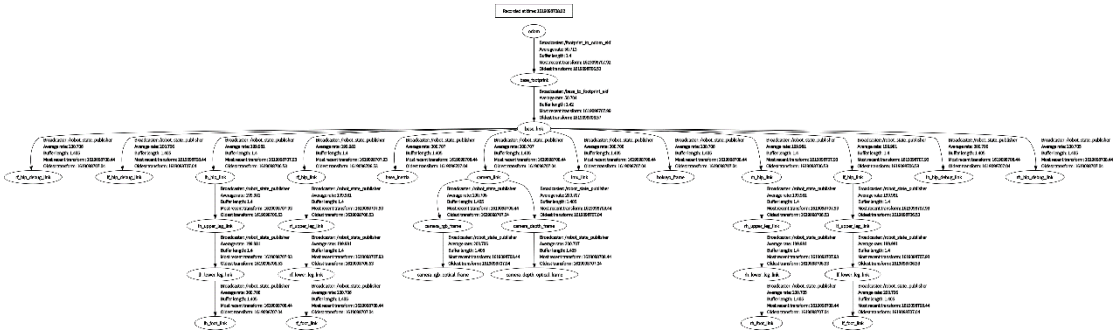


Figure 6.6 The graph of TF tree when the two launch files are running

CHAPTER 7 Conclusion & Recommendation

7.1 Conclusion

There are 3 objectives done but one objective failed in this project. The following content will talk about those objectives in detail.

7.1.1 Learning kinematics algorithm

From CHAPTER 2 and CHAPTER 5, the algorithm about forward and inverse kinematics that will be applied in the robot has been addressed in math and program. Therefore, this objective has been achieved. However, for the kinematics programming, it is cloned from Github. Even though the math algorithm about forward and inverse kinematics has been studied, the kinematics programming is not original.

7.1.2 Modelling the robot in SolidWorks

From CHAPTER 4 the objective of modelling is addressed with the design sketch. Therefore, this objective has been achieved.

7.1.3 Learning how to use ROS

The main idea of this project is to use ROS to achieve the function of a quadruped robot. So nearly the whole report is talking about ROS. The basic knowledge about ROS and some part of programming skills about ROS are addressed in this report. Therefore, this objective has been achieved. However, the ROS package is cloned from GitHub, because it is too difficult to program the whole project without a high level of programming skill.

7.1.4 Controlling the real quadruped robot

This project failed to achieve this objective. Even though the owner of the project tried to laser cut and 3D print the robot model, due to the rough sketch and lack of manufacturing experience, the real robot model failed to be assembled. Another problem is that when the project owner trying to control the motor by using Arduino and raspberry pi. The system failed to be controlled by ROS. Although it has linked successfully with ROS master, the message can’t be published or received by nodes. Due to the limited time, this part is failed.

7.2 Recommendation

For students who want to do similar project in the future, here are some recommendations.

1. ROS is a good system for program robot, because there are lots of open-sourced project on github that can be grabbed free. However, for mechanical engineering student, who don't have much background in linux, ros may not be the first choice for program robot. Because the learning investment is too large.
2. Matlab is a good tool for mechanical engineering students to program robot, because we are more familiar with it ,and there are more tutorials about matlab than ros.

References

1. JIMENO, Juan Miguel. *chvmp_champ*.

2. LIU, Jing, TAN, Min and ZHAO, Xiaoguang. Legged robots — an overview. *Transactions of the Institute of Measurement & Control*. 2007. Vol. 29, no. 2, p. 185–202. DOI 10.1177/0142331207075610.

3. PYO, Yoonseok, CHO, Hancheol, RYUWOON, Jung and LIM, Taehoon. *ROS Robot Programming*. 2017.

4. ROS. *ROS.org* [online]. 2020. Available from: https://www.ros.org/about-ros/

5. BOSTON DYNAMICS. *Spot® | Boston Dynamics* [online]. 2019. Available from: https://www.bostondynamics.com/spot

6. SYNDICATE MARKET RESEARCH. *Quadruped Robot Market By Type (Spider Shape, Dog Shape, Others), By Application (Rescue, Military, Industry, Others), and By Region - Overall In-depth Analysis, Global Market Share, Top Trends, Professional & Technical Industry Insights 2020 - 2026*. 2020.

7. ANYBOTICS. *ANYmal C – Autonomous Legged Robot | ANYbotics* [online]. 2019. Available from: https://www.anybotics.com/anymal-legged-robot/

8. HASAN, Khan Saad Bin. *What, Why and How of ROS - Towards Data Science* [online]. 2020. Available from: https://towardsdatascience.com/what-why-and-how-of-ros-b2f5ea8be0f3

9. ROBERT, John. *Hands-On Introduction to Robot Operating System(ROS) _ trojrobert* [online]. 2020. Available from: https://trojrobert.github.io/hands-on-introdution-to-robot-operating-system%28ros%29/

10. MATHWORKS. *Simscape - MATLAB & Simulink* [online]. 2020. Available from: https://www.mathworks.com/products/simscape.html?s_tid=srchtitle%0Ahttps://www.mathworks.com/products/simscape.html

11. MATLAB. *ROS Toolbox* [online]. 2020. Available from: https://de.mathworks.com/products/ros.html

12. MATHWORKS. *Get Started with Gazebo and a Simulated TurtleBot* [online]. 2019. Available from: https://uk.mathworks.com/help/robotics/examples/get-started-with-gazebo-and-a-simulated-turtlebot.html

13. KUCUK, Serdar and BINGUL, Zafer. *Robot Kinematics: Forward and Inverse Kinematics*. 2006.

14. UBUNTU. *Enterprise Open Source and Linux | Ubuntu* [online]. 2020. Available from: https://ubuntu.com/

15. RASPBERRY PI. *Teach, learn and make with raspberry pi* [online]. 2018. Available from: https://www.raspberrypi.org/The Raspberry Pi is a tiny and affordable computer that you can use to learn programming through fun, practical projects. Join the global Raspberry Pi Community.

16. ZHOU, Todd. *Raspberry Pi Ubuntu20.04-arm64 test and setting - SCUTEEE.pdf* [online]. 2020. Available from: https://scuteee.com/2020/04/27/raspberry-pi-Ubuntu2004-arm64-setup.html

17. GUYUEJU. *ROS2 beginner tutorial* [online]. 2020. Available from: https://www.guyuehome.com/10226

18. CAD GRAB. *GrabCAD: Design Community, CAD Library, 3D Printing Software* [online]. 2020. Available from: grabcad.com

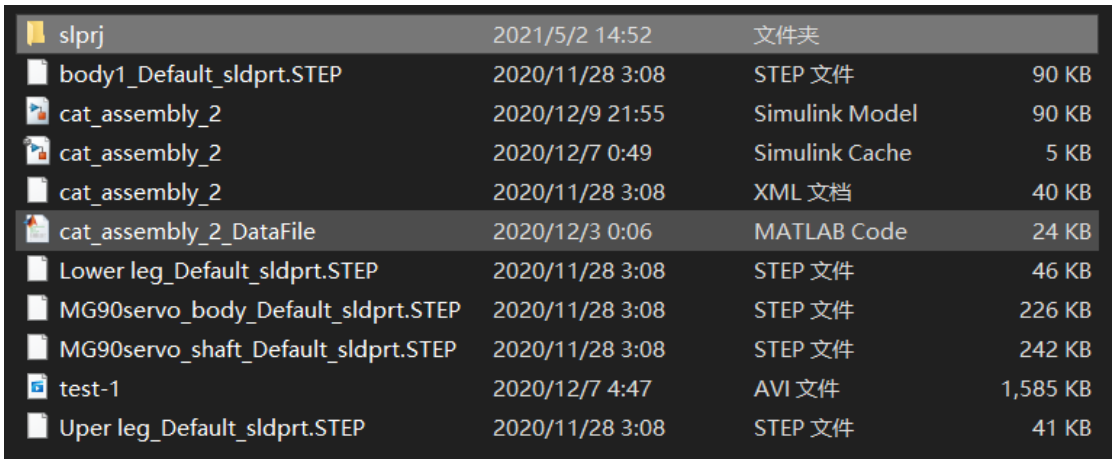
19. JAMES. *GitHub - XRobots_openDog_ CAD and code for each episode of my open source dog series* [online]. 2020. Available from: https://github.com/XRobots/openDog#start-of-content

Appendix

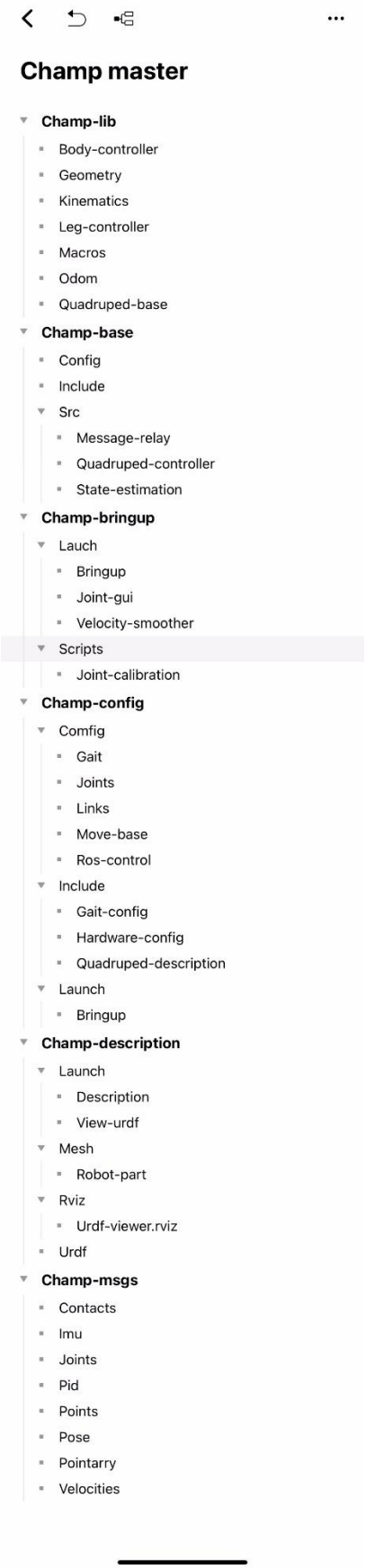
3D Model of the robot

Robot_body
Robot_lower_leg_1
Robot_lower_leg_2
Robot_lower_leg_3
Robot_lower_leg_4
Robot_upper_leg_1
Robot_upper_leg_2
Robot_upper_leg_3
Robot_upper_leg_4
Mg90_servo*8

Matlab Simulink



ROS champ package



ROS teleop

Telpoperation.launch

Champ_teleop.py