YEAR 2 PROJECT

Simulating Quadrupedal Locomotion in PyBullet

Minghong XU (ID 201601082)

Kai YANG (ID 201458046)

Zehao YE (ID 201601167)

Zepeng PANG (ID 201600756)

Group 2p42

Supervised by Dr Murat UNEY

March 19, 2022

Abstract

This report first introduces the background knowledge and objective of the quadrupedal locomotion simulation project. The objective of the project is divided into four parts, corresponding to four simulations of motions, which are pitching, yawing, rolling, and squatting motion. Various software and plug-ins used in this project, such as simulation platform pybullet and VSCode editor, are introduced respectively afterwards. Then, the method of modeling the legs of a quadruped robot is explained in detail. Subsequently explained how to use trigonometry knowledge and kinematic algorithms to derive kinematic models for each motion. The report concludes by presenting the results of the project, with the simulation of all motions successfully completed. Meanwhile, we discussed intellectual property, future work, reflection, and other aspects about this project.

In addition, the code in the project, project management forms, and individual contributions of each member will be provided in the Appendix section. respectively.

Declaration

I confirm that I have read and understood the University's definitions of plagiarism and collusion from the Code of Practice on Assessment. I confirm that I have neither committed plagiarism in the completion of this work nor have I colluded with any other party in the preparation and production of this work. The work presented here is my own and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web). I understand the consequences of engaging in plagiarism and collusion as described in the Code of Practice on Assessment (Appendix L).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

There are currently three types of mobile robots: wheeled robots, tracked robots, and footed robots. Compared with the other two robots, the footed robot has outstanding advantages. For example, on natural terrain, a footed robot can easily move on sand, hard or soft ground with similar efficiency [1]. It can still move even when it encounters discontinuous terrain. Wheeled robots can move efficiently only on flat and hard surfaces. Although tracked robots have certain off-road performance, their efficiency in harsh terrain conditions is still not as good as that of footed robots. In addition, In addition, in terms of slippage and jamming, since the actions of stepping on and raising legs of the footed robot are perpendicular to the ground and will not interfere with the ground, the footed robot can easily walk on soft and muddy roads [1]. In contrast, the turning of the wheels of a wheeled robot causes the body to sink and become difficult to move. Therefore, the research on footed robots is a hot topic in the current scientific field.

The motion of the legs of a footed robot is quite complex, and in this project we refer to the method proposed by Denavit and Hartenberg in 1955. This method uses triangular relations and kinematics algorithms to derives the kinematic model [1].

1.2 Objectives

The objective of this project is to complete the simulation of the posture adjustment motions of the quadruped robot on the Pybullet platform. The motions simulated in the three directions are pitch, roll, and yaw motion. The project will develop programs for three actions respectively. Finally, three programs

should be combined to complete the posture adjustment of the quadruped robot on three axes in space. In addition, the height adjustment of the robot can be regarded as a part of the posture adjustment. The height adjustment program is combined into the posture adjustment program to achieve the additional goal of adjusting the posture of the quadruped robot at any height.

Chapter 2

Materials and Methods

2.1 Materials

2.1.1 Simulation Enviornment

- **programming language:** Python 3.10
- **robotics simulator:** PyBullet 3.2.0
- **quadruped robot 3D model:** Unitree A1
- **packages that extend Python's array programming capability:**
  Numpy 1.22.2
  Pandas 1.4.0

2.1.2 Development Environment

- **terminal emulator:** Windows Terminal
- **shell:** PowerShell 7.2.1
- **package manager:** Scoop
- **Python version manager:** pyenv-win
- **Python packaging and dependency manager:** Poetry
- **version control system:** Git
- **code hosting platform:** GitHub

- **languages for documentation:**
  - Markdown
  - LaTeX 2$_\varepsilon$
    - TeX distribution: MiKTeX 21.12
    - typesetting engine: XeTeX
    - reference management: BibTeX
  - Mermaid 8.14.0
- **editor:** VSCode
  - **Python support:** Python extension pack 2022.2.1924087327
  - **Markdown support:** Markdown All in One 3.4.0
  - **LaTeX support:** LaTeX Workshop 8.23.0
- **3D CAD modeling:** Shapr3D
- **file format:** EditorConfig
- **blog framework:** Hexo
  - **theme:** NexT
  - **Markdown renderer:** hexo-renderer-marked
  - **MathJax renderer:** hexo-filter-mathjax
- **static site hosting service:** GitHub Pages

2.2 Methods

2.2.1 Modelling

Firstly, a geometric model of the quadruped robot was established.

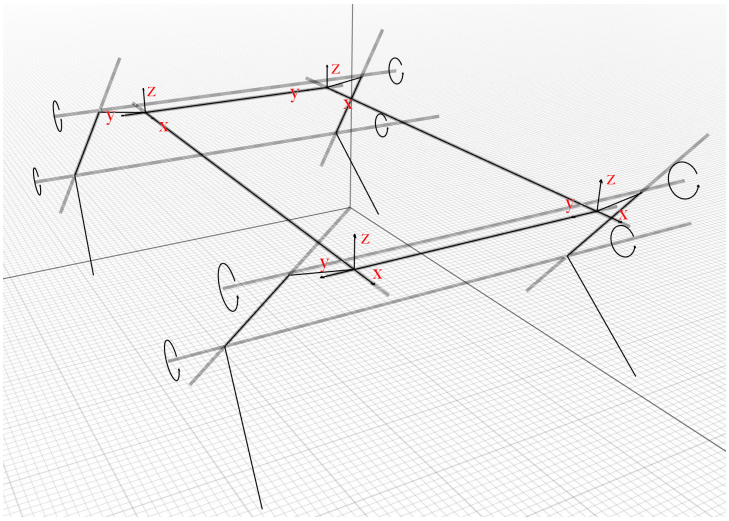

Figure 2.1: Geometric model of the quadruped robot

Figure 2.1 shows a trimetric view of the model. The body is reduced to a square and the legs are reduced to three connecting rods. The joint between the body and the legs is called hip. The uppermost rod of the leg is called the hip offset, followed by the thigh and finally the shank. In a robot entity, the distance from centre of motor controlling hip abduction/adduction to top centre of the thigh is approximated by the hip offse

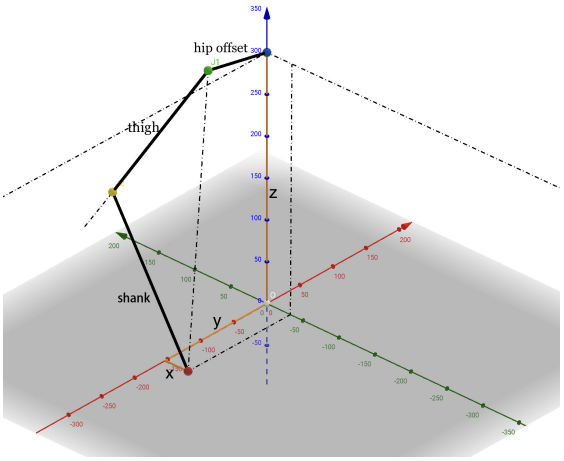

Figure 2.2: Right leg model

The leg has four important parts. There are three movable parts, named

hip abduction/adduction (abd/add) joint, hip flextion/extension (fle/ext) joint, and knee joint from top to bottom. The end of the leg is called toe.

In order to describe leg movements using mathematics, it is first necessary to establish a coordinate system. In terms of leg movements, the hip and the toe are the parts of the leg that receive the most attention; because once the position of the toe in relation to the hip has been determined, the shape of the leg is fixed. The hip is located in the body and is a immovable point for the body, so setting it as the origin is convient for thinking. The positive direction of the coordinate system is shown in Figure 2.1. There will be a coordinate system on each hip. This means that the four legs will be studied separately.

Specifying initial position of the motor and positive direction of rotation is as important as establishing the coordinate system. The quadruped robot model used for this project, Unitree A1, specifies them in its unified-robotics-description-format file, so these provisions are directly followed. The initial position of the motors and the positive direction of rotation is shown in Figure 2.3.

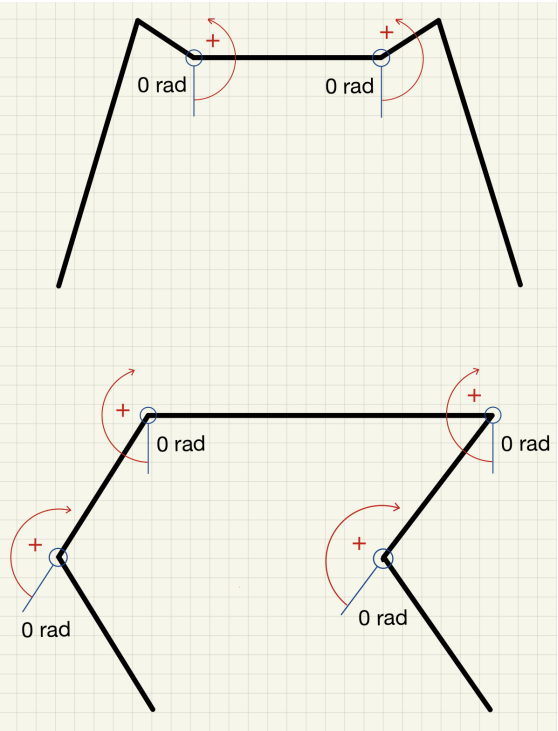

Figure 2.3: Initial position and positive direction of motors

The Unitree A1's unified-robotics-description-format file also specifies the boundary positions of the motors. They are listed in Table 2.1. For the sake of readability, front-left is abbreviated as fl, front-right as fr, hind-left as hl, and hind-right as hr.

The next step is to derive the relationship between the toe coordinates before and after posture adjustment. The posture adjustment including pitching, yawing, rolling, and squatting.

Table 2.1: Motors boundary positions

Motors	Positive Boundary (rad)	Negative Boundary (rad)
fl hip abd/add	0.803	-0.803
fr hip abd/add	0.803	-0.803
hl hip abd/add	0.803	-0.803
hr hip abd/add	0.803	-0.803
fl hip fle/ext	4.189	-1.047
fr hip fle/ext	4.189	-1.047
hl hip fle/ext	4.189	-1.047
hr hip fle/ext	4.189	-1.047
fl knee	-0.916	-2.697
fr knee	-0.916	-2.697
hl knee	-0.916	-2.697
hr knee	-0.916	-2.697

Pitching

When pitching, the body length is constant, and the central axis of the body remains in the same position. Using these two conditions, the coordinates after pitching can be obtained by transforming the coordinate system three times. Take the front-right leg as an example:

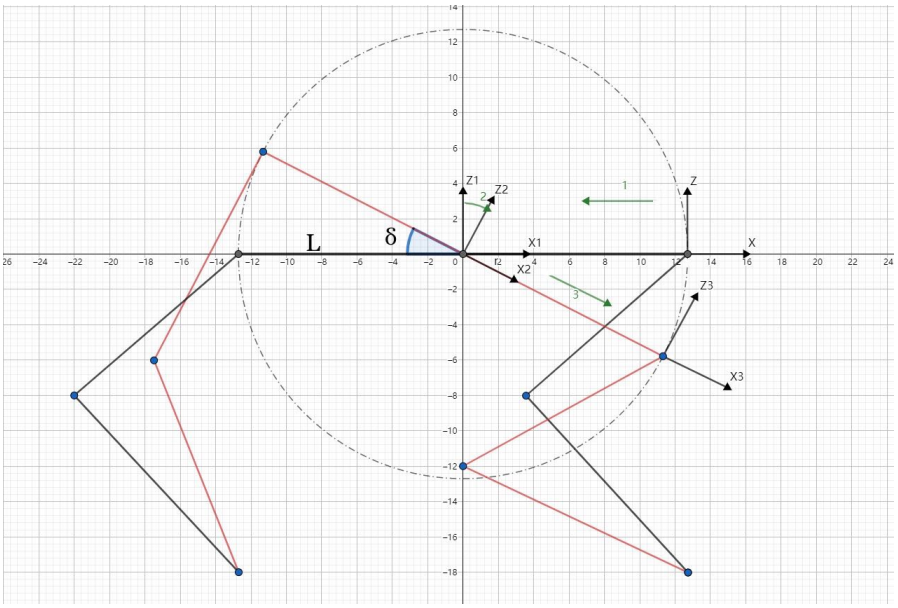

Figure 2.4: Coordinate transformations in pitching

The process of translating the coordinate system from hip to the central axis can be described by pre-multipling a transformation matrix:

$$\begin{bmatrix} x_{\text{after translating}} \\ z_{\text{after translating}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & L \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \\ 1 \end{bmatrix} \tag{2.1}$$

where L is half body length.

Then, rotate the coordinate system by the pitching angle δ and transform it to the new location of the hip. The matrix multiplication for this process is

$$\begin{bmatrix} x_{\text{after pitching}} \\ z_{\text{after pitching}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\delta & -\sin\delta & -L \\ \sin\delta & \cos\delta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{after translating}} \\ z_{\text{after translating}} \\ 1 \end{bmatrix} \tag{2.2}$$

The coordinates after pitching are obtained. Note that the y-coordinate does not change when pitching.

The two two matrix multiplication above can be combined into one

$$\begin{bmatrix} x_{\text{after pitching}} \\ z_{\text{after pitching}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\delta & -\sin\delta & L \times \cos\delta - L \\ \sin\delta & \cos\delta & L \times \sin\delta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \\ 1 \end{bmatrix} \tag{2.3}$$

The above calculation process is all for the front-right leg. By calculating the other legs it can be seen that for both **front** legs Eq 2.3 holds. For **hind** legs, the equation is

$$\begin{bmatrix} x_{\text{after pitching}} \\ z_{\text{after pitching}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\delta & -\sin\delta & -L \times \cos\delta + L \\ \sin\delta & \cos\delta & -L \times \sin\delta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ z \\ 1 \end{bmatrix} \tag{2.4}$$

Adding the information that y does not change during pitching to the matrix multiplication, the final form for the **front** legs is

$$\begin{bmatrix} x_{\text{after pitching}} \\ y_{\text{after pitching}} \\ z_{\text{after pitching}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\delta & 0 & -\sin\delta & L \times \cos\delta - L \\ 0 & 1 & 0 & 0 \\ \sin\delta & 0 & \cos\delta & L \times \sin\delta \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.5}$$

and for the **hind** legs is

$$\begin{bmatrix} x_{\text{after pitching}} \\ y_{\text{after pitching}} \\ z_{\text{after pitching}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\delta & 0 & -\sin\delta & -L \times \cos\delta + L \\ 0 & 1 & 0 & 0 \\ \sin\delta & 0 & \cos\delta & -L \times \sin\delta \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.6}$$

Yawing

The above idea also applies to yawing. One difference is that the matrix for translating each of the four coordinate systems is different, so that ultimately each leg corresponds to a matrix. See Figure 2.5.

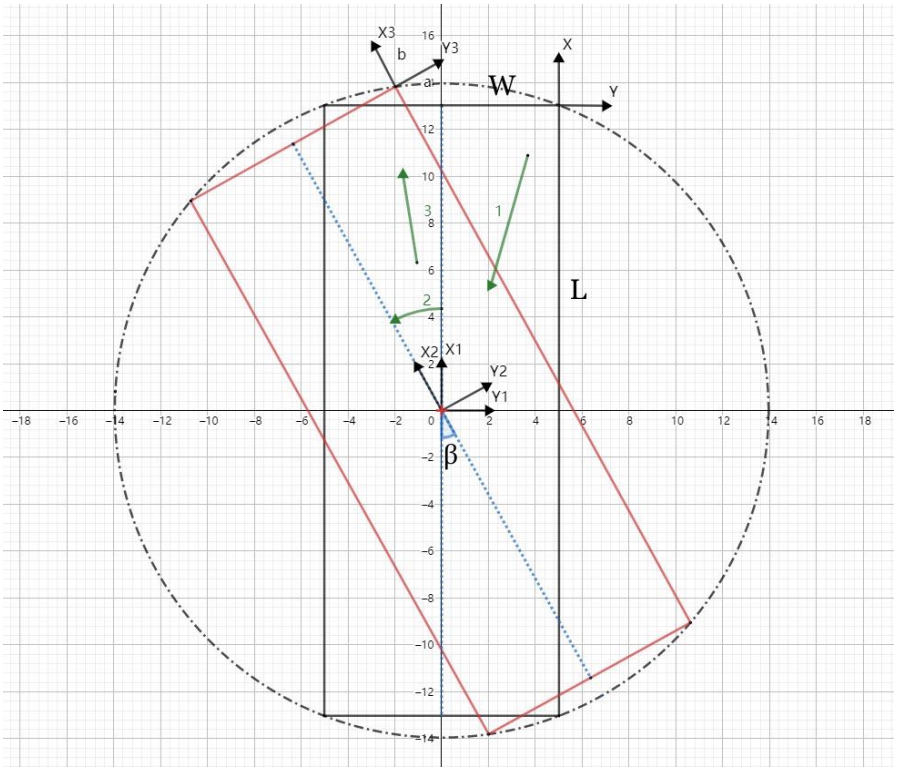

Figure 2.5: Coordinate transformations in yawing

As the derivation has already been described in section Pitching, it is omitted here, and the final form of matrix multiplication for yawing for each leg are listed directly. Give that the yawing angle is β, half body length is L, and half body width is W.

For **front-right** leg:

$$\begin{bmatrix} x_{\text{after yawing}} \\ y_{\text{after yawing}} \\ z_{\text{after yawing}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 & L\times\cos\beta - W\times\sin\beta - L \\ \sin\beta & \cos\beta & 0 & L\times\sin\beta + W\times\cos\beta - W \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.7}$$

For **front-left** leg:

$$\begin{bmatrix} x_{\text{after yawing}} \\ y_{\text{after yawing}} \\ z_{\text{after yawing}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 & L\times\cos\beta + W\times\sin\beta - L \\ \sin\beta & \cos\beta & 0 & L\times\sin\beta - W\times\cos\beta + W \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.8}$$

For **hind-right** leg:

$$\begin{bmatrix} x_{\text{after yawing}} \\ y_{\text{after yawing}} \\ z_{\text{after yawing}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 & -L\times\cos\beta - W\times\sin\beta + L \\ \sin\beta & \cos\beta & 0 & -L\times\sin\beta + W\times\cos\beta - W \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.9}$$

For **hind-left** leg:

$$\begin{bmatrix} x_{\text{after yawing}} \\ y_{\text{after yawing}} \\ z_{\text{after yawing}} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 & -L \times \cos\beta + W \times \sin\beta + L \\ \sin\beta & \cos\beta & 0 & -L \times \sin\beta - W \times \cos\beta + W \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.10}$$

Rolling

Unlike yawing, in rolloing, the right two legs share a transformation matrix, as do the left two legs. This is because the translation operation of the coordinate system is different for the left and right sides, while the rotation operation is the same for each leg.

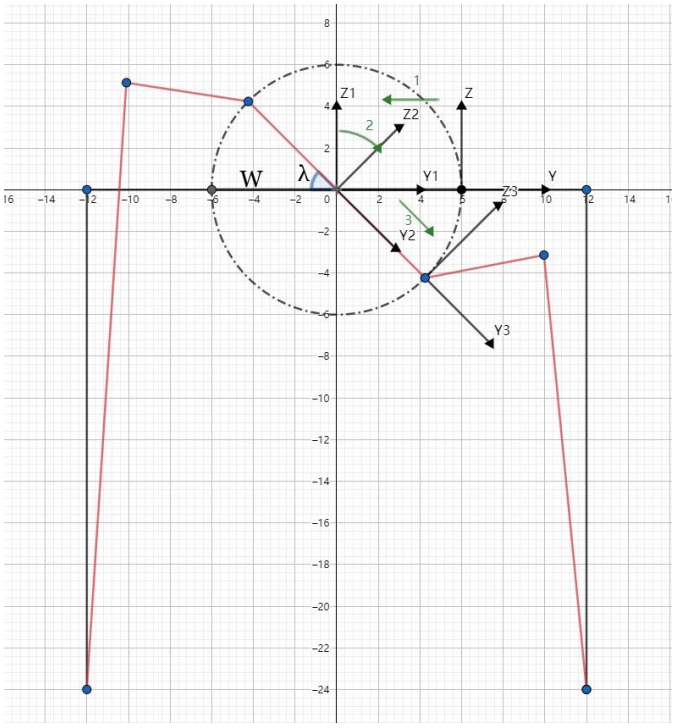

Figure 2.6: Coordinate transformations in rolling

Give that rolling angle is λ, and half body width is W. Again, the derivation is omitted and only the final form of matrix multiplication is presented.

For **right** legs:

$$\begin{bmatrix} x_{\text{after rolling}} \\ y_{\text{after rolling}} \\ z_{\text{after rolling}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\lambda & -\sin\lambda & W \times \cos\lambda - W \\ 0 & \sin\lambda & \cos\lambda & W \times \sin\lambda \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.11}$$

For **left** legs:

$$\begin{bmatrix} x_{\text{after rolling}} \\ y_{\text{after rolling}} \\ z_{\text{after rolling}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\lambda & -\sin\lambda & -W \times \cos\lambda + W \\ 0 & \sin\lambda & \cos\lambda & -W \times \sin\lambda \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.12}$$

Squatting

Squatting is actually the process of continuously changing the height of each hip above the ground. In all cases, the z-coordinate represents this height. Assuming that the standing surface does not change, then simply changing the z-coordinate can achieve a squat.

2.2.2 Forward and Inverse Kinematics

In the previous section, the transformation matrix of the coordinates of the toe after posture adjustment was obtained. However, only the motor position can be directly controlled, and also the robot's controller can only return information about the motor position and not the coordinates of the toe relative to the hip joint. Therefore, in order to control the motor position to achieve posture adjustment, both forward and inverse kinematics are required.

The forward kinematics means finding the coordinates of toe relative to the hip given motors position which is servo angle, while inverse kinematics is the opposite. This conversion relation can be derived elegantly using several mathematical concepts such as Euler angle. However, as it is too far beyond the knowledge of the team, the choice here is to use basic geometric knowledge to describe this relationship.

It is necessary to discuss the left and right sides in separate cases, as the Unitree A1's right-side motor configuration of the hip abd/add joint which includes initial position and positive direction does not consistent with the left side. Again, to save space, the equations on both sides are given directly.

Give that thigh length is l_1, shank length is l_2, distance from the toe to the top point of the thigh rod is h, position of the hip fle/ext joint is θ_1, and position of the knee joint is θ_2. The following equation can be obtained easily from the left-side view of the right leg model shown in Figure 2.7.

$\theta_1, \theta_2 \rightarrow x, h$:

$$\begin{aligned} x &= -l_1 \times \sin\theta_1 - l_2 \times \sin(\theta_1 + \theta_2) \\ h &= l_1 \times \cos\theta_1 + l_2 \times \cos(\theta_1 + \theta_2) \end{aligned}$$

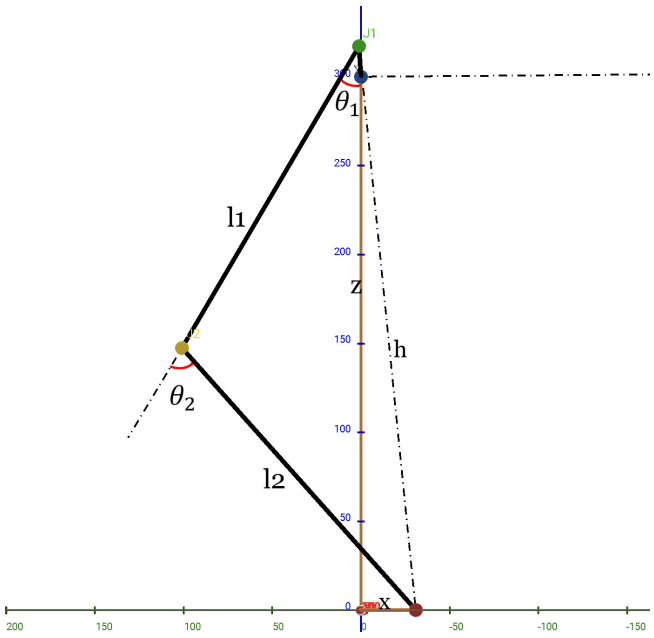

Figure 2.7: Left-side view of the right leg model

$x, h \rightarrow \theta_1, \theta_2$:

$$\cos\theta_2 = \frac{-l_1^2 - l_2^2 + x^2 + h^2}{2l_1 l_2}$$

$$\sin\theta_2 = -\sqrt{1 - \cos^2\theta_2}$$

$$\theta_1 = \arccos\left(\frac{l_1^2 + x^2 + h^2 - l_2^2}{2l_1\sqrt{x^2 + h^2}}\right) - \arctan2\left(\frac{x}{h}\right)$$

$$\theta_2 = \arctan2\left(\frac{\sin\theta_2}{\cos\theta_2}\right)$$

Give that the hip offset is o and position of hip abd/add joint is θ_3. A second set of equations can be obtained from the front view of the right leg model (Figure 2.8). As mentioned above, the motor configuration of hip abd/add joint is different on the left and right side; hence, two sets of equations are required.

For the **right** side:

$\theta_0 \rightarrow y, z$:

$$y = o \times \cos\theta_0 - h \times \sin\theta_0$$

$$z = -o \times \sin\theta_0 - h \times \cos\theta_0$$

$y, z \rightarrow \theta_0$:

$$\theta_0 = \arctan2\left(\frac{|z|}{y}\right) - \arctan2\left(\frac{h}{o}\right)$$

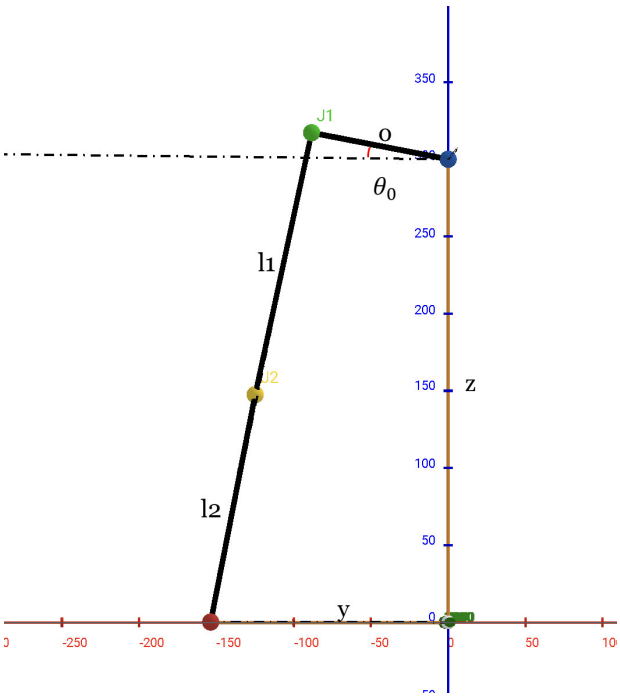

Figure 2.8: Front view of the right leg model

but for the **left** side:

$\theta_0 \rightarrow y, z$:

$$y = -o \times \cos\theta_0 - h \times \sin\theta_0$$
$$z = o \times \sin\theta_0 - h \times \cos\theta_0$$

$y, z \rightarrow \theta_0$:

$$\theta_0 = \text{arctan2}(\frac{h}{o}) - \text{arctan2}(\frac{|z|}{-y})$$

Summary

Forward kinematics:

$\theta_1, \theta_2 \rightarrow x, h$:

$$x = -l_1 \times \sin\theta_1 - l_2 \times \sin(\theta_1 + \theta_2) \quad (2.13)$$
$$h = l_1 \times \cos\theta_1 + l_2 \times \cos(\theta_1 + \theta_2) \quad (2.14)$$

$\theta_0 \rightarrow y, z$ for **right** side:

$$y = o \times \cos\theta_0 - h \times \sin\theta_0 \quad (2.15)$$
$$z = -o \times \sin\theta_0 - h \times \cos\theta_0 \quad (2.16)$$

$\theta_0 \rightarrow y, z$ for **left** side:

$$y = -o \times \cos\theta_0 - h \times \sin\theta_0 \tag{2.17}$$
$$z = o \times \sin\theta_0 - h \times \cos\theta_0 \tag{2.18}$$

Inverse kinematics:

$x, h \rightarrow \theta_1, \theta_2$:

$$\cos\theta_2 = \frac{-l_1^2 - l_2^2 + x^2 + h^2}{2l_1 l_2} \tag{2.19}$$
$$\sin\theta_2 = -\sqrt{1 - \cos^2\theta_2} \tag{2.20}$$
$$\theta_1 = \arccos(\frac{l_1^2 + x^2 + h^2 - l_2^2}{2l_1\sqrt{x^2 + h^2}}) - \arctan2(\frac{x}{h}) \tag{2.21}$$
$$\theta_2 = \arctan2(\frac{\sin\theta_2}{\cos\theta_2}) \tag{2.22}$$

$y, z \rightarrow \theta_0$ for **right** side:

$$\theta_0 = \arctan2(\frac{|z|}{y}) - \arctan2(\frac{h}{o}) \tag{2.23}$$

$y, z \rightarrow \theta_0$ for **left** side:

$$\theta_0 = \arctan2(\frac{h}{o}) - \arctan2(\frac{|z|}{-y}) \tag{2.24}$$

2.2.3 Coding

Equation derivation and coding were carried out simultaneously. After a set of equations for a movement had been derived, a simple Python script was used to verify correctness. If the result seen in the simulator was correct, the algorithm in the script would then be encapsulated into a library.

Once a movement had been ecapsulated, a corresponding application was written for debugging and testing purpose. After all the movements completed, a demo application was written which used in presentation of bench inspection.

Chapter 3

Results and Analysis

The results of the project demonstrate the locomotion of quadrupedal robot, which are pitching, yawing, rolling, and squatting. By adjusting parameters through corresponding custom sliders, a series of actions are obtained through screenshots and merged into figures, where certain text interpretation would be executed. Meanwhile, the analysis would contribute to the comparison among states of joints. The motions of the quadrupedal robot would be dissected respectively.

3.1 Squatting

Squatting, for the quadruped robot, is the action that makes the gravity centre of its main body rise or fall vertically. By pulling the custom slider of squatting to adjust the parameters, a series of actions of the quadruped robot when squatting is shown in Fig.3.1.

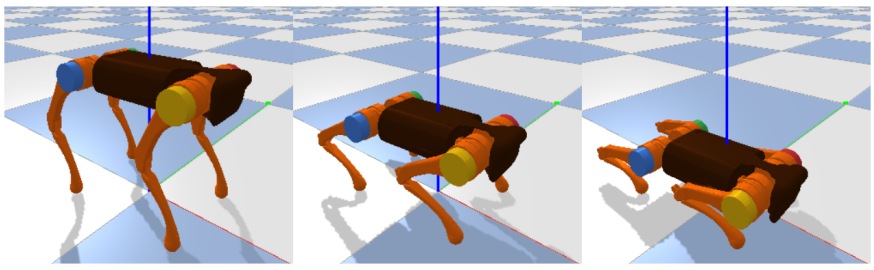

Figure 3.1: Squatting

In the initial position, the distance from hip joints to toe joints is 270. While the slider of debug parameter was pulled to the left, the distance became smaller, the body of the robot descended vertically. While the slider of debug parameter was pulled to the right, the distance became larger, and the robot's body ascended vertically.

The state of abduction/adduction hip joints would retain the initial state while squatting. The state of flection/extension hip joints and knee joints would change corresponding to the distance from the toe. Judging from the simulation results, the robot's legs were crooked while the robot's trunk moves downward, and were extended while the robot's trunk move upward. The state of knee joints. The state of the knee joints determined the current highest position of the robot, and then the actual position of the robot was adjusted by controlling the state of the hip joint.

3.2 Pitching

Pitching, for the quadruped robot, is the action of rotating about the transverse axis [2]. By pulling the custom slider of pitching to adjust the parameters, a series of actions of the quadruped robot when pitching is shown in Fig.3.2.

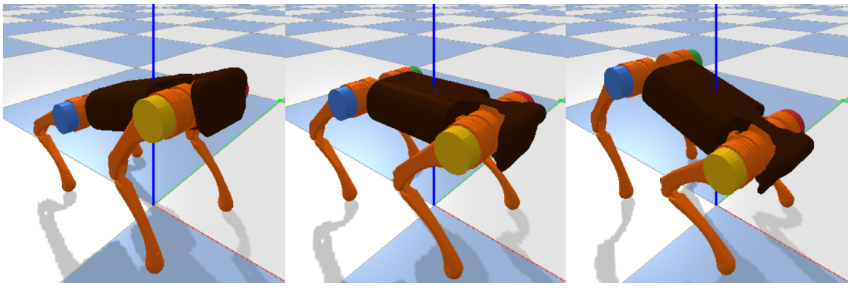

Figure 3.2: Pitching

In the initial position, the body of the quadruped robot remained horizontal, which means that the pitch angle was 0. While the slider of debug parameter was pulled to the left, the pitch angle became negative, and the robot's body made counterclockwise rotation about the transverse axis. While the slider of debug parameter was pulled to the right, the pitch angle became positive, and the robot's body made clockwise rotation about the transverse axis.

As mentioned in the method, the motion of the robot body only occurs in the plane perpendicular to the transverse axis. Therefore, the variations of joints' states between the left and right legs are consistent, unilateral legs need to be evaluated in the analysis. Compared to the motions of yawing and rolling, the motion of pitching is more straightforward, because the state of abduction/adduction hip joints would retain the initial state. However, in the process of pitching, the state of flection/extension hip joints and knee joints would change with the flection or extension of the legs. While the legs were extended, the calves rotated clockwise, from the left perspective, relative to the knee joints., the thighs rotated counterclockwise, from the left perspective, relative to the flection/extension hip joints. While the legs were crooked, the calves rotated counterclockwise, from the left perspective, relative to the knee joints, the thighs rotated clockwise, from the left perspective, relative to the flection/extension hip joints.

3.3 Yawing

Yawing, for the quadruped robot, is the action of rotating about the normal axis [2]. By pulling the custom slider of squatting to adjust the parameters, a series of actions of the quadruped robot when yawing are shown in Fig.3.3.

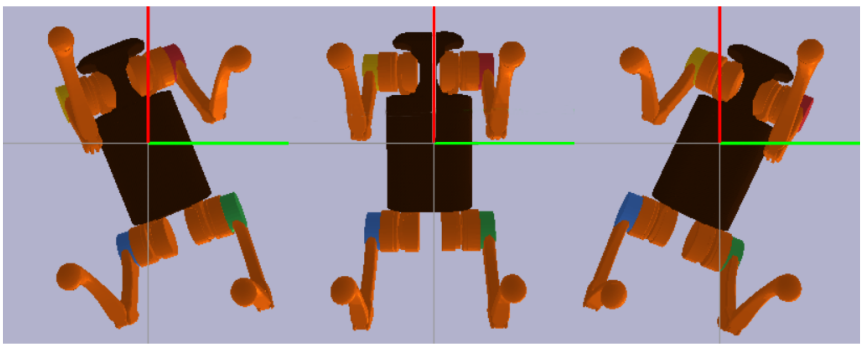

Figure 3.3: Yawing

For the sake of observing the locomotion of the quadruped robot more intuitively, we chose the bottom view as the observation view in the screenshots of yawing. In the initial position, the body of the quadruped robot was parallel to the longitudinal axis, which means that the yaw angle was 0. While the slider of debug parameter was pulled to the left, the yaw angle became negative, and the robot's body made counterclockwise rotation about the normal axis. While the slider of debug parameter was pulled to the right, the yaw angle became positive, and the robot's body made clockwise rotation about the normal axis.

It can be seen that the leg state of the robot is diagonally symmetrical, that is, the joints' state of the front-left leg is similar to that of the hind-right leg, and the joints' state of the front-right leg is similar to that of the hind-left leg. While the robot yawing to right, the front legs rotated counterclockwise, and the hind legs rotated clockwise, from the front perspective, relative to the abduction/adduction hip joints.The front-right leg and hind-left leg were crooked. The front-left leg and hind-right leg were extended. The state of these joints is actually similar to the motion of leg joints in squatting.

3.4 Rolling

Rolling, for the quadruped robot, is the action of rotating about the longitudinal axis [2]. By pulling the custom slider of squatting to adjust the parameters, a series of actions of the quadruped robot when rolling are shown in Fig.3.4.

For the sake of observing the locomotion of the quadruped robot more intuitively, we chose the front view as the observation view in the screenshots of

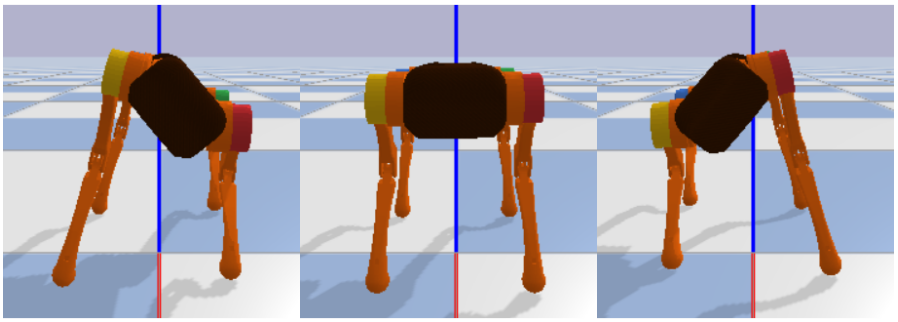

Figure 3.4: Rolling

rolling. In the initial position, the heights of the left and right hip joints from the ground are the same, which means that the roll angle was 0. While the slider of debug parameter was pulled to the left, the roll angle became negative, and the robot's body made clockwise rotation about the longitudinal axis. While the slider of debug parameter was pulled to the right, the roll angle became positive, and the robot's body made counterclockwise rotation about the longitudinal axis.

The change in the joints' state of rolling is similar to pitching, except that the leg state of the robot is left and right symmetrical. While the robot pitched clockwise, the front legs rotated counterclockwise, and the hind legs rotated clockwise, from the front perspective, relative to the abduction/adduction hip joints. The right-side legs would be extended and the left-side legs would be crooked.

Chapter 4

Discussion

4.1 Limitations and Improvement

4.1.1 Ranges of Debug Parameters

In the simulation, the debug parameters were utilized to adjust the locomotion of the quadruped robot. The ranges of debug parameters are given in advance. The fixed ranges would not affect the robot when it performs a single action. Nevertheless, it is not appropriate for the robot to use the fixed range when performing multiple actions at the same time. For example, if the robot stretches its legs to reach the highest squatting position, which means the legs have reached the longest. Afterwards, moving other sliders will lead to program errors, and the robot will disappear in the GUI[3] owing to the errors.

To avoid the error caused by the fixed range, the approach is to calculate the real-time debug parameter range according to the joint state of the robot. However, the real-time value ranges need to get the joints' state firstly, then bring them into the correlation matrix to deduce the value range inversely, and compare it with the value range of the joint itself. The complexity of real-time range is close to the quadrupedal locomotion, so it is hard for us to spare time to optimize this part of the code. Otherwise, there is another way to improve the robustness of the code, which is to narrow the range make it so that the robot cannot reach the limit range. Although the problem is not solved from the root, there is less possibility that the code will make mistakes during operation.

4.1.2 Non-zero steady-state error

When the robot is stationary, its joints are not completely static. The robot's joints would move in a minute range. The range of this steady-state error is approximately five decimal places. However, the current state read is used as

a reference when controlling the attitude of the robot. Therefore, the previous steady-state error will be substituted into the next action, and the program does not know the generation of this error. In the process of research, we find that as long as the accuracy of the parameters is controlled in three places after the decimal point, the steady-state error can be effectively reduced. But in theory, the problem of steady-state error should not be solved in this way. Therefore, in the following aspects that can be improved, we hope to reduce the steady-state error by using the robust integral controller[4] for the nonlinear system.

4.1.3 Friction between the plane and robot

When dragging the slider rapidlly to adjust the attitude of the robot, the robot will shift from the original position. This is because the friction between the toe joints of the robot and the plane is too small to provide the necessary force for the robot to move under this acceleration. The solution is to increase the friction coefficient between the toe joints and the plane in order to increase the friction force.

4.2 Future work

Due to lack of time, many different adaptations, tests, and experiments are left for the future (the depth of research that can be done in four weeks is limited). Future work can involve in-depth analysis of the motion mechanism of quadruped robots and try more simulations of actions. This project mainly focuses on three actions of the quadruped robot, namely pitch angle, yaw angle, and roll angle, and also includes a squat action. The basic theories are basically from robotics and mathematics (geometry and linear algebra), and the following ideas can be tested in future work:

4.2.1 New motions

1. It may be interesting to consider more new motions, such as Z-axis-based bouncing or climbing, depending on deep learning and flexible use of robotics, such as creating new kinematic equations and matrices. For example, using these movements in practice will facilitate movement on more complex terrain. In addition, try to study more complex movements, such as back flips, which Boston Dynamics achieved.

2. Create a new robot model (Unified Robot Description Format): Instead of using an existing open-source model, the robot is described based on the XML syntax framework. ROS provides a C++ interpreter for URDF files, which can

parse this kind of robot. format file. Fortunately, although starting from building a model is a tedious task, it is possible to start with what we have learned in the second semester of ELEC230.

4.3 Intellectual property

This project uses the robot model ”A1” of Shuyu Technology, which is a patent belonging to their company. It is necessary to declare their patent here: U.S. patent number D916,202 [Application Number D/718,514] was granted by the patent office on 2021-04-13 for quadruped robot. The grantee listed for this patent is HangZhou YuShu TECHNOLOGY CO., LTD.. Invention is credited to Xingxing Wang, Zhiyu Yang. »»»> Stashed changes

4.4 Completion of objectives

4.4.1 Pitching

When the pitch angle is 0, the body of the robot remains perpendicular to the longitudinal axis perpendicular to the ground. By changing the pitch angle, the front or rear leg of the robot is flexed, while the other leg is extended. What is visually reflected is that the robot is constantly leaning forward and backward with the adjustment of the pitch angle. The simulation of the pitching motion was quite successful.

4.4.2 Yawing

When changing the magnitude of the yaw angle, it can be seen from the direction perpendicular to the body of the robot from below that the legs on the front side of the robot move in opposite directions to the legs on the back side. The body of the robot rotates during yaw. When the yaw angle is positive, the robot body rotates clockwise, and when the yaw angle is negative, the robot body rotates counterclockwise. However, due to the influence of ground friction, the robot will slide when the yaw angle is changed many times, but this does not affect the simulation. The objective for the simulation of yaw motion was also successfully completed.

4.4.3 Rolling

When changing the magnitude of the roll angle, from the front view of the robot, the leg on one side of the robot is extended and the leg on the other side is flexed. The observed scenario is that one side of the robot is raised in height, while the other side is lowered in height. This means that the simulation of the roll motion has been successfully completed.

4.4.4 squatting

As an additional goal of the project, squatting is actually simpler than the other three motions, so the simulation of this action is done first. When we pull the height slider in the program, the four legs of the robot will flex or stretch at the same time, thus completing the adjustment of the body height.

4.4.5 Motion combination

After combining the three simulation programs, an attitude control program that can adjust three angles at the same time is obtained. In Fig.4.1, the first picture is to adjust the roll angle, the second picture is to adjust the pitch angle based on the first picture, and the third picture is to adjust the yaw angle based on the second picture.

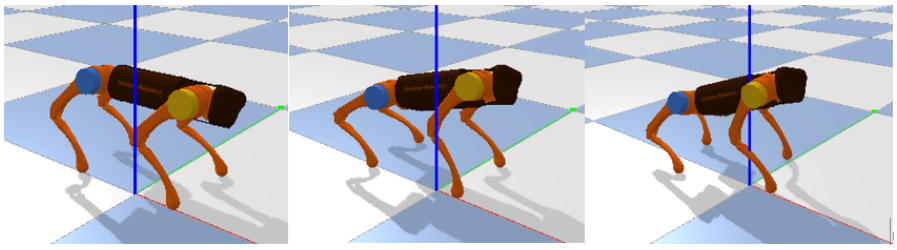

Figure 4.1: Combination of three motions

After combining the attitude adjustment program including three actions with the height adjustment program, the attitude adjustment program with height adjustment function is obtained, which is the final result of this project. The simulation results are shown in Fig.4.2. The first picture in Fig.4.2 is the attitude of the robot after adjusting the pitch angle, yaw angle and roll angle. By adjusting the height, the attitude in the second picture is obtained.

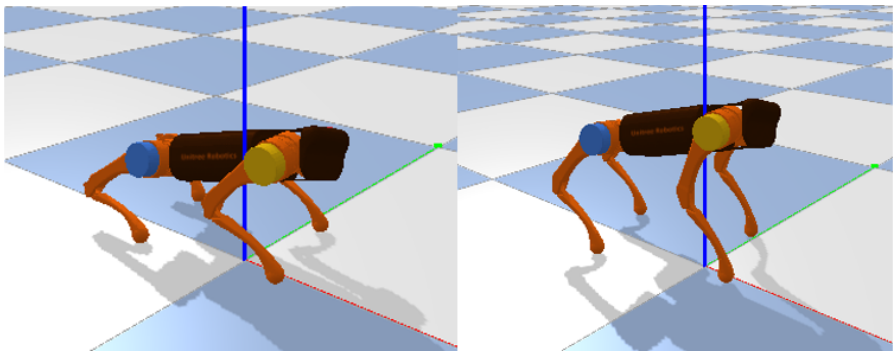

Figure 4.2: Pose control with height

Chapter 5

Conclusion

This report explains the development ideas and final results of the year 2 project. First the project was split into three main motion simulations, pitch, yaw and roll motions and a secondary goal, height adjustment. The ultimate objective is to design a robot attitude control program with height control function. Then we introduced the development environment of the project. Next we explained how to model the legs of a quadruped robot and use trigonometry knowledge and forward and inverse kinematics algorithms to deduce the kinematic model of the quadruped robot legs. After that we showed the simulation results of height adjustment, pitch, yaw and roll motions respectively. We then discussed this pose control program in terms of limitations and improvement, future work, intellectual property, and completion of objectives. Hopefully this year 2 project can be helpful to students who choose quadruped robot simulation as their subject in the future.

References

[1] P. G. de Santos, E. Garcia, and J. Estremera, *Quadrupedal locomotion: an introduction to the control of four-legged robots*. Springer, 2006.

[2] “Six degrees of freedom,” Apr 2020. [Online]. Available: https://en.wikipedia.org/wiki/Six_degrees_of_freedom

[3] “Graphical user interface,” Aug 2019. [Online]. Available: https://en.wikipedia.org/wiki/Graphical_user_interfac

[4] Z.-P. Jiang and I. Marcels, “Robust nonlinear integral control,” *IEEE Transactions on Automatic Control*, vol. 46, no. 8, pp. 1336–1342, 2001.

Appendices

Appendix A

Project management forms

Y2 project (ELEC222/ELEC273) - *Attendance record*

Notes:

- ***This sheet should be updated by group members weekly when they meet to discuss the project.***
- ***Assessors will request to see this sheet in the bench inspection day.***

	Member name	Attended the weekly meeting? (Yes/No)					Comments
		Week 1	Week 2	Week 3	Week 4	Week 5	
1	Minghong Xu	Yes	Yes	Yes	Yes		full work attendance
2	Zehao Ye	Yes	Yes	Yes	Yes		full work attendance
3	Kai Yang	Yes	Yes	Yes	Yes		full work attendance
4	Zepeng Pang	Yes	Yes	Yes	Yes		full work attendance
5							

Y2 project (ELEC222/ELEC273) – *Role allocation (responsibility matrix)*

Notes:

- ***Assessors will request to see this sheet in the bench inspection day.***
- ***See overleaf for details on titles and associated roles and responsibilities.***

	Member Name	**Title(s)**
1	Minghong Xu	Project Manager, Designer, and Implementer
2	Zehao Ye	Designer and Implementer
3	Kai Yang	Implementer and Technical Writer
4	Zepeng Pang	Technical Writer and Implementer
5		

Responsibility Matrix

KEY:
R – Responsible (accountable) for completion of task. (Task can be delegated to this person.)
S – Supports task.
C – Requires communication about the task.

Title	Project Activity				Deliverables			
	Requirements/ Scope	Design	Implementing	Testing	Poster	Blog	Bench	Report
Project Manager	R	C	S	S	S	S	S	S/R
Designer	C	R	S	S	S	S	S	S
Implementer/ Developer	C	S	R	S	C	C	R	S
Technical Writer	C	S	S	S	R	R	C	R

Typical Roles

Title	Role	Responsibilities
Project Manager	Responsible for developing, in conjunction with the supervisor, the project scope. The Project Manager ensures that the project is delivered on time and to the required standards.	• Managing and lead the project team. • Managing the coordination of the partners and the working groups.
Designer	Designing the system (the circuit, the code, etc.).	• Creating the required block diagrams, circuit diagrams, flow charts, etc.
Implementer/Developer	Implementing the suggested design	• Connecting the systems/circuit • Writing the code
Technical Writer	Documenting the project progress and deliverables	• Recording and maintaining all meeting logs • Updating the logbook. • Creating project blog • Creating project poster • Writing project report
<Title>	<Role>	• <Responsibility>
<Title>	<Role>	• <Responsibility>
<Title>	<Role>	• <Responsibility>
<Title>	<Role>	• <Responsibility>

Y2 project (ELEC222/ELEC273) - *Contribution to project deliverables*

Notes:

- ***Assessors will request to see this sheet in the bench inspection day.***
- ***Typical deliverables include: Bench, Poster, Blog, Report, Code, Circuit, etc.***

	Member name	Deliverable(s)	Comments
1	Minghong Xu	simulation code of every motion, attendance record	Excellent completion of all assigned tasks.
2	Zehao Ye	simulation code of pitching and yawing, poster	Excellent completion of all assigned tasks.
3	Kai Yang	week 1 and week 2 blogs, simulation code of rolling, poster, sustainability report	Excellent completion of all assigned tasks.
4	Zepeng Pang	supervisor weekly meeting log, week 1~4 blogs, motion code assembles	Excellent completion of all assigned tasks.
5			

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 1*

Date: 31.1.2022 Supervisor: Dr Murat Uney

Project Title: Simulation of quadruped robot locomotion

Student Names /Attendees:	1. Minghong Xu	2. Kai Yang
3. Zehao Ye	4. Zepeng Pang	5.

Summary of week's activities:

At the Supervisor meeting on Monday, the group determined that the task of the first week was to complete the construction of the robot simulation environment for each member. With Minghong's help, the rest of the team successfully completed the construction of the pybullet simulation environment and other preparations on Thursday. In detail, we started virtual environment by using python-poetry. We also installed LaTex, Markdown and Python extensions in VSCode to be prepared for future poster, report and code work. Additionally, we configured Git and learned its basic usage. Finally we preliminarily explored the application of pybullet in robot motion simulation.

Problem, issues and concerns:

Group members need to be more familiar with how to use pybullet in the simulation of robot motion. The robot model required for the simulation could not be determined. Project schedule has started to lag behind expectations.

Tasks for next week/Actions for next meeting:

Task for everyone: Investigate how to simulate robot dog movement using pybullet.

Other tasks:

Minghong Xu/ Kai Yang: Write the weekly blog.

Zepeng Pang: Write the supervisor weekly meeting log.

Supervisor use only

Progress Assessment: □ Unsatisfactory ☒ Satisfactory □ Good

Comments/Recommendations:

Planning should decompose the end goal into SMART tasks each associated with one person in the group.

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 2*

Date: 7.2.2022 Supervisor: Dr Murat Uney

Project Title: Simulation of quadruped robot locomotion

Student Names /Attendees:	1. Minghong Xu	2. Kai Yang
3. Zehao Ye	4. Zepeng Pang	5.

Summary of week's activities:

At supervisor's suggestion, the group designed a gantt chart to make work arrangement clear to everyone. In the lab on Thursday, we determined the robot model to use for simulation. After importing the model into pybullet, everyone successfully attempted to write code to simply move the model's joints. In addition, in order to write programs more conveniently, we installed the Jupyter plugin on VSCode. Finally, We studied the kinematic modeling of the legs of a quadruped robot and the attitude adjustment matrix, which is also based on kinematic.

Problem, issues and concerns:

It's a bit hard to understand the knowledge of kinematic modeling. Team members need to continue learning about simulating robots with pybullet.

Tasks for next week/Actions for next meeting:

Task for everyone: Try to complete a simple simulated movement.

Other tasks:

Minghong Xu/ Kai Yang: Write the weekly blog.

Zepeng Pang: Write the supervisor weekly meeting log.

Supervisor use only

Progress Assessment: □ Unsatisfactory □ Satisfactory ☒ Good

Comments/Recommendations: Team has started to demonstrate good skills in setting realistic goals, requirement specifications and planning.

Tasks should be continually revised for realistic end times and goals.

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 3*

Date: 14.2.2022 Supervisor: Dr Murat Uney

Project Title: Simulation of quadruped robot locomotion

Student Names /Attendees:	1. Minghong Xu	2. Kai Yang
3. Zehao Ye	4. Zepeng Pang	5.

Summary of week's activities:

At the supervisor meeting on Monday, The team decided to simulate three movements: pitch motion, yaw motion, and roll motion. In the knowledge of the kinematic modeling of the legs of the quadruped robot studied last week, we learned the knowledge of pitch angle, yaw angle and roll angle. The knowl edge of these three angles corresponds to the three motion we want to simulate. The team decided th at three people would each undertake a simulation of a motion, and the remaining one would be resp onsible for the paperwork which contains blog, supervisor weekly meeting log and preparation of the p oster.

Problem, issues and concerns:

Group members need to keep learning commands and functions which would be used in pybullet. Sp end more time in the last two weeks to speed up the project.

Tasks for next week/Actions for next meeting:

Minghong Xu: Complete the simulation of robot's roll motion.

Zehao Ye: Complete the simulation of robot's pitch motion.

Kai Yang: Complete the simulation of robot's yaw motion.

Zepeng Pang: Complete paperwork (blogs, meeting log, preparation of poster).

Supervisor use only

Progress Assessment: □ Unsatisfactory □ Satisfactory ☑ Good

Comments/Recommendations: The team has gained good traction in revising plans to balance the end-goal with the time/labour budget.

The demonstration scripts for different motion modes should be prepared before they are finished - one person can undertake this task in addition to other responsibilities.

Year 2 Project (ELEC222/273) – *Supervisor meeting – Week 4*

Date: 21.2.2022 Supervisor: Dr Murat Uney

Project Title: Simulation of quadruped robot locomotion

Student Names /Attendees:	1. Minghong Xu	2. Zehao Ye
3.	4.	5.

Summary of week's activities:

After last week's research, we successfully deduced the matrix transformation formulas of pitch, yaw and roll angles this week. On this basis, the code of pitch, yaw and roll actions was developed, and the simulation of the three actions of the quadruped robot was successfully completed. The main tasks of the project have been completed. After that, we started designing the poster and preparing for next week's bench inspection.

Problem, issues and concerns:

The poster has not yet been designed, which may take up our time on Saturday and Sunday. The members responsible for the paperwork may not be familiar with the code.

Tasks for next week/Actions for next meeting:

1. Design the poster and make it
2. Prepare documents required for bench inspection
3. Everyone in the group should be familiar with the code

Supervisor use only

Progress Assessment: □ Unsatisfactory □ Satisfactory ☑ Good

Comments/Recommendations: So far the team has demonstrated a very good level of project planning and execution in a student defined project. A good presentation in bench inspection and a well written report is strongly recommended.

Appendix B

Individual contributions to the project

B.1 Minghong Xu

Although I was involved in most of the work, I was mainly responsible for modelling, programming, and project management. I derived the transformation matrices for yawing and pitching and implemented the simulation of these two movement in code. I revised the equations for forward and inverse kinematics derived by Zehao Ye and correct the transformation matrix for rolling which was derived by Kai Yang. In addition, I had also implemented a simple squatting simulation and a program which control each motor individually for visualisation. As a technician, I set up the infrastructure for the project, including but not limited to the blog site, GitHub repository, Python and LaTeXdevelopment environment, and specified the toolchain and workflow. As I am familiar with the toolchain, I am responsible for teaching others how to use these tools. When someone had a problem with a tool, I helped them to solve it. As a project manager, I set weekly goals, maintained the GitHub repository, and a Gantt chart.

B.2 Zehao Ye

In this group project, I am mainly responsible for three aspects, which are quadrupedal locomotion simulation, presentation and report. From the perspective of quadrupedal locomotion simulation, I have completed the forward and inverse kinematics derivation and matrix derivation of some parts of pitch, yaw and roll actions. The part I participated in most was pitching. Meanwhile, debug parameter section was mainly designed by me. From the perspective of presentation, I drew the poster through software. In the poster, except that the pictures of the modelling part were made by Yang Kai, other parts were completed by me. In presentation,

I responsed for the forward and inverse kinematics of quadrupedal locomotion. From the perspective of report, I wrote the results and analysis section. In addition, the limitations of the discussion and conclusion section are also completed by me.

B.3 Zepeng Pang

At the beginning of the project, I was responsible for the writing of the supervisor weekly meeting log and Blog. Later responsible for modifying other project management forms such as attendance record and role allocation. After the group completed the simulation of the four independent actions, I combined the four programs into a single program and added the ability to initialize the viewing angle to the program. In bench inspection, I explained the project introduction and how to model the legs of the quadruped robot. In the writing of the report, I was responsible for the writing of the title page, abstract, introduction and conclusion. In addition, I also completed the Completion of objectives of the discussion chapter and Appendix A.

B.4 Kai Yang

In this group project, I was involved in quadruped motion simulation, cartography, and report writing. Based on robotics and trigonometry, I completed the kinematic derivation and code for the roll angle part. The images used in the posters and reports of this group are all made based on professional geometric software, which can well represent the characteristics of the model. In the final report, I completed the future work outlook and intellectual property statement. In addition, I put forward a lot of ideas in the SDE report.

Appendix C

Code

C.1 Library for Unitree A1 3D model

```
import pybullet as bullet
import pandas as pd
import numpy as np
from enum import Enum
from typing import Iterable

class A1:

    class EnumWithListMethod(Enum):
        @classmethod
        def list(cls):
            return list(map(lambda c: c.value, cls))

    class Leg(EnumWithListMethod):  # FIXME: Use 3.11 enum.StrEnum to remove .value
        fr = 'front-right'
        fl = 'front-left'
        hr = 'hind-right'
        hl = 'hind-left'

    class Motor(EnumWithListMethod):  # FIXME: Use 3.11 enum.StrEnum to remove .value
        h_aa = 'hip abduction/adduction'
        h_fe = 'hip flexion/extension'
        knee = 'knee'

    motor_indices = pd.DataFrame(dtype=np.int8, index = Motor.list(), columns = Leg.list(),
                        # front-right, front-left, hind-right, hind-left
        data = np.array([[     1     ,      6    ,     11    ,     16   ],    # hip abduction/adduction
                         [     3     ,      8    ,     13    ,     18   ],    # hip flexion/extension
                         [     4     ,      9    ,     14    ,     19   ]]))  # knee
    # Obtained by measuring the STL file of the thigh and calf. Both are 0.2mm.
    thigh_len  = 200
    shank_len  = 200
    hip_offset = 80   # FIXME: Obtain more accurate value
    body_len   = 360  # FIXME: Obtain more accurate value
    body_width = 200  # FIXME: Obtain more accurate value


    def __init__(
    self,
    physics_client_id: int,
    base_position: Iterable[float] = [0, 0, 0.43],
    base_orientation: Iterable[float] = [0, 0, 0, 1]):
        self.in_physics_client = physics_client_id
        # Load the model in the given PyBullet physics client.
        import pybullet_data; bullet.setAdditionalSearchPath(pybullet_data.getDataPath())  # The A1 model is in the
        ↪  pybullet_data
        self.id = bullet.loadURDF(
            "a1/a1.urdf",
            base_position, base_orientation,
            physicsClientId = self.in_physics_client,
            flags = bullet.URDF_USE_SELF_COLLISION,
            useFixedBase = False)
        # Set initial postion of motors.
        bullet.setJointMotorControlArray(
            physicsClientId = self.in_physics_client,
            bodyUniqueId = self.id,
            jointIndices = self.motor_indices.loc[A1.Motor.h_aa.value, :],  # FIXME: Use 3.11 enum.StrEnum to
            ↪  remove .value
            controlMode = bullet.POSITION_CONTROL,
```

```
            targetPositions = np.full(4, 0),)
        bullet.setJointMotorControlArray(
            physicsClientId = self.in_physics_client,
            bodyUniqueId = self.id,
            jointIndices = self.motor_indices.loc[A1.Motor.h_fe.value, :],  # FIXME: Use 3.11 enum.StrEnum to
            ↪  remove .value
            controlMode = bullet.POSITION_CONTROL,
            targetPositions = np.full(4, 0.7),)
        bullet.setJointMotorControlArray(
            physicsClientId = self.in_physics_client,
            bodyUniqueId = self.id,
            jointIndices = self.motor_indices.loc[A1.Motor.knee.value, :],  # FIXME: Use 3.11 enum.StrEnum to
            ↪  remove .value
            controlMode = bullet.POSITION_CONTROL,
            targetPositions = np.full(4, -0.7*2),)  # ensures that the toes are beneath the hips


    def motor_info(self):
        """
        A matrix contains motor information. The info is structured as
        Tuple(jointIndex, jointName, jointType, qIndex, uIndex, flags, jointDamping,
        jointFriction, jointLowerLimit, jointUpperLimit, jointMaxForce, jointMaxVelocity,
        linkName, jointAxis, parentFramePos, parentFrameOrn, parentIndex)

        See PyBullet document for getJointInfo().
        """
        return A1.motor_indices.applymap(lambda index: bullet.getJointInfo(self.id, index))


    def current_pose(self):
        return A1.motor_indices.applymap(lambda index: bullet.getJointState(self.id, index,
        ↪  self.in_physics_client)[0])


    def pose_control(self, roll_angle=0., pitch_angle=0., yaw_angle=0., Δz=0., reference_pose=None) -> None:
        """
        Adjust pose of fuselage; move the fuselage from a reference placement to a desired placement

        fuselage
        : the central body portion of an aircraft designed to accommodate the crew and the passengers or cargo
        Available: https://www.merriam-webster.com/dictionary/fuselage [Accessed: 17 March 2022]

        Attitude and position fully describe how an object is placed in space. (For some applications such as in
        robotics and computer vision, it is customary to combine position and attitude together into a single
        description known as Pose.)
        Available: https://en.wikipedia.org/wiki/Attitude_control#Geometry [Accessed: 17 March 2022]
        Further Read: https://en.wikipedia.org/wiki/Pose_(computer_vision)

        In geometry, the orientation, angular position, attitude, or direction of an object such as a line, plane,
        or rigid body is part of the description of how it is placed in the space it occupies. More specifically,
        it refers to the imaginary rotation that is needed to move the object from a reference placement to its
        current placement. A rotation may not be enough to reach the current placement. It may be necessary to add
        an imaginary translation, called the object's location (or position, or linear position). The location and
        orientation together fully describe how the object is placed in space. The above-mentioned imaginary
↪  rotation
        and translation may be thought to occur in any order, as the orientation of an object does not change when
        it translates, and its location does not change when it rotates.

        Euler's rotation theorem shows that in three dimensions any orientation can be reached with a single
↪  rotation
        around a fixed axis. This gives one common way of representing the orientation using an axis-angle
        representation. Other widely used methods include rotation quaternions, rotors, Euler angles, or rotation
        matrices. More specialist uses include Miller indices in crystallography, strike and dip in geology and
↪  grade
        on maps and signs. Unit vector may also be used to represent an object's normal vector orientation.

        Typically, the orientation is given relative to a frame of reference, usually specified by a Cartesian
        coordinate system.

        The attitude of a rigid body is its orientation as described, for example, by the orientation of a frame
        fixed in the body relative to a fixed reference frame. The attitude is described by attitude coordinates,
↪  and
        consists of at least three coordinates. One scheme for orienting a rigid body is based upon body-axes
        rotation; successive rotations three times about the axes of the body's fixed reference frame, thereby
        establishing the body's Euler angles. Another is based upon roll, pitch and yaw, although these terms also
        refer to incremental deviations from the nominal attitude.

        The attitude is described by attitude coordinates, and consists of at least three coordinates.

        Available: https://en.wikipedia.org/wiki/Orientation_(geometry) [Accessed: 17 March 2022]

        Tait-Bryan angles is the convention normally used for aerospace applications, so that zero degrees
↪  elevation
        represents the horizontal attitude. Tait-Bryan angles represent the orientation of the aircraft with
↪  respect
        to the world frame. When dealing with other vehicles, different axes conventions are possible.

        Alternative names
        For an aircraft, they can be obtained with three rotations around its principal axes if done in the proper
        order. A yaw will obtain the bearing, a pitch will yield the elevation and a roll gives the bank angle.
        Therefore, in aerospace they are sometimes called yaw, pitch and roll. Notice that this will not work if
↪  the
        rotations are applied in any other order or if the airplane axes start in any position non-equivalent to
↪  the
        reference frame.
```

```
    Available: https://en.wikipedia.org/wiki/Euler_angles [Accessed: 17 March 2022]

    :param Δz:
        Height is not easily defined for a legged robot on uneven terrains; hence, z-coordinate of the toe
        relative to the coordinate system on the hip of each leg is used.
    :param reference_pose:
        A frame of reference which the attitude and position given by the other params is relative to
    """

    if reference_pose is None:
        # FIXME: Taking an angle of 0.0 and calculating it repeatedly with respect to the current pose,
        # it can be observed that the errors add up and the pose of the robot is slowly distorted.
        # Possible Sol: Decrease to 3 decimal places
        pose = self.current_pose()
    else:
        pose = reference_pose.copy()

    for leg, θ in pose.items():  #! Pass by reference; modify θ will also modify the element in pose
        # θ is a tuple of three motor angular positions of one leg

        # Abbreviating to increase readability
        # FIXME: Adhere to typical notation (ψ, θ, φ) given by the Euler angles
        λ = roll_angle
        δ = pitch_angle
        β = yaw_angle
        L = A1.body_len / 2
        W = A1.body_width / 2

        x, y, z = A1._forward_kinematics(leg, *θ)

        # Adjust height
        z -= Δz  # FIXME: Wrong algrithm.

        # x, y, z after rolling
        match leg:
            case A1.Leg.fr.value | A1.Leg.hr.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                roll = np.array([[ 1,  0,          0,          0                ],
                                 [ 0,  np.cos(λ), -np.sin(λ),  W * np.cos(λ) - W ],
                                 [ 0,  np.sin(λ),  np.cos(λ),  W * np.sin(λ)     ],
                                 [ 0,  0,          0,          1                ]])
            case A1.Leg.fl.value | A1.Leg.hl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                roll = np.array([[ 1,  0,          0,          0                ],
                                 [ 0,  np.cos(λ), -np.sin(λ), -W * np.cos(λ) + W ],
                                 [ 0,  np.sin(λ),  np.cos(λ), -W * np.sin(λ)     ],
                                 [ 0,  0,          0,          1                ]])
        x, y, z, _ = roll.dot(np.array([x, y, z, 1]))

        # x, z after pitching
        match leg:
            case A1.Leg.fr.value | A1.Leg.fl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                pitch = np.array([[ np.cos(δ), -np.sin(δ),  L * np.cos(δ) - L ],
                                  [ np.sin(δ),  np.cos(δ),  L * np.sin(δ)     ],
                                  [ 0,          0,          1                 ]])
            case A1.Leg.hr.value | A1.Leg.hl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                pitch = np.array([[ np.cos(δ), -np.sin(δ), -L * np.cos(δ) + L ],
                                  [ np.sin(δ),  np.cos(δ), -L * np.sin(δ)     ],
                                  [ 0,          0,          1                 ]])
        x, z, _ = pitch.dot(np.array([x, z, 1]))

        # x, y, z after yawing
        match leg:
            case A1.Leg.fr.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                yaw = np.array([[ np.cos(β), -np.sin(β),   0,   L * np.cos(β) - W * np.sin(β) - L ],
                                [ np.sin(β),  np.cos(β),   0,   L * np.sin(β) + W * np.cos(β) - W ],
                                [ 0,          0,           1,   0                                 ],
                                [ 0,          0,           0,   1                                 ]])
            case A1.Leg.fl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                yaw = np.array([[ np.cos(β), -np.sin(β),   0,   L * np.cos(β) + W * np.sin(β) - L ],
                                [ np.sin(β),  np.cos(β),   0,   L * np.sin(β) - W * np.cos(β) + W ],
                                [ 0,          0,           1,   0                                 ],
                                [ 0,          0,           0,   1                                 ]])
            case A1.Leg.hr.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                yaw = np.array([[ np.cos(β), -np.sin(β),   0,  -L * np.cos(β) - W * np.sin(β) + L ],
                                [ np.sin(β),  np.cos(β),   0,  -L * np.sin(β) + W * np.cos(β) - W ],
                                [ 0,          0,           1,   0                                 ],
                                [ 0,          0,           0,   1                                 ]])
            case A1.Leg.hl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                yaw = np.array([[ np.cos(β), -np.sin(β),   0,  -L * np.cos(β) + W * np.sin(β) + L ],
                                [ np.sin(β),  np.cos(β),   0,  -L * np.sin(β) - W * np.cos(β) + W ],
                                [ 0,          0,           1,   0                                 ],
                                [ 0,          0,           0,   1                                 ]])
        x, y, z, _ = yaw.dot(np.array([x, y, z, 1]))

        θ[0], θ[1], θ[2] = A1._inverse_kinematics(leg, x, y, z)

        # FIXME: If θ[0] or θ[1] or θ[2] exceeds its limits, aborting them by using return

    # Control angular position of each motor
    for positions_of_one_leg, indices_of_one_leg in zip(pose.itertuples(index=False),
    ↪  A1.motor_indices.itertuples(index=False)):
        bullet.setJointMotorControlArray(
            physicsClientId = self.in_physics_client,
            bodyUniqueId = self.id,
            controlMode = bullet.POSITION_CONTROL,
```

```
            jointIndices = indices_of_one_leg,
            targetPositions = positions_of_one_leg,)


    @classmethod
    def _forward_kinematics(cls, leg: Leg, θ0, θ1, θ2):
        """
        θ0, θ1, θ2 ⇒ x, y, z

        :param θ0: angular position of hip abd/add motor
        :param θ1: angular position of hip fle/ext motor
        :param θ2: angular position of knee motor
        :return  : a tuple, (x, y, z), which is coordinates of the toe relative to the hip
        """

        # Abbreviating to increase readability
        l1 = cls.thigh_len
        l2 = cls.shank_len
        o  = cls.hip_offset
        h  =  l1 * np.cos(θ1) + l2 * np.cos(θ1 + θ2)  # distance from the toe to the top centre of the thigh rod

        x = -l1 * np.sin(θ1) - l2 * np.sin(θ1 + θ2)
        match leg:
            case A1.Leg.fr.value | A1.Leg.hr.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                y =  o * np.cos(θ0) - h * np.sin(θ0)
                z = -o * np.sin(θ0) - h * np.cos(θ0)
            case A1.Leg.fl.value | A1.Leg.hl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                y = -o * np.cos(θ0) - h * np.sin(θ0)
                z =  o * np.sin(θ0) - h * np.cos(θ0)

        return (x, y, z)


    @classmethod
    def _inverse_kinematics(cls, leg: Leg, x, y, z):
        """
        x, y, z ⇒ θ0, θ1, θ2

        :param x, y, z: coordinates of the toe relative to the hip
        :return       : a tuple, (θ0, θ1, θ2), which is angular position of three motors on a leg
        """

        # Abbreviating to increase readability
        l1 = cls.thigh_len
        l2 = cls.shank_len
        o  = cls.hip_offset
        h  = np.sqrt(z**2 + y**2 - o**2)

        cosθ2 = (-l1**2 - l2**2 + x**2 + h**2) / (2 * l1 * l2)
        sinθ2 = -np.sqrt(1 - cosθ2**2)  # takes negative suqare root, because the knee's position is specified as
        ↪  negative
        match leg:
            case A1.Leg.fr.value | A1.Leg.hr.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                θ0 = np.arctan2(np.abs(z), y) - np.arctan2(h, o)
            case A1.Leg.fl.value | A1.Leg.hl.value:  # FIXME: Use 3.11 enum.StrEnum to remove .value
                θ0 = np.arctan2(h, o) - np.arctan2(np.abs(z), -y)
        θ1 = np.arccos((l1**2 + x**2 + h**2 - l2**2) / (2 * l1 * np.sqrt(x**2 + h**2))) - np.arctan2(x, h)
        θ2 = np.arctan2(sinθ2, cosθ2)

        return (θ0, θ1, θ2)
```

C.2 Application for demonstration

```
import pybullet as bullet
from liba1 import A1
from time import sleep

# Initialisation
physics_server_id = bullet.connect(bullet.GUI)
bullet.setRealTimeSimulation(enableRealTimeSimulation=True, physicsClientId=physics_server_id)
bullet.setGravity(0, 0, -30, physics_server_id)
import pybullet_data; bullet.setAdditionalSearchPath(pybullet_data.getDataPath())
bullet.loadURDF("plane.urdf", physicsClientId=physics_server_id)
a1 = A1(physics_server_id)
bullet.resetDebugVisualizerCamera(
    physicsClientId = physics_server_id,
    cameraTargetPosition = [0, 0, 0.4],
    cameraDistance = 1.5,
    cameraYaw = 40,
    cameraPitch = -15)

# Debug parameter silders to adjust the robot's motions
debug_roll_angle = bullet.addUserDebugParameter(
    paramName = "roll angle (rad)",
    rangeMin = -0.5,
    rangeMax =  0.5,
    startValue = 0,)
```

```
debug_pitch_angle = bullet.addUserDebugParameter(
    paramName = "pitch angle (rad)",
    rangeMin = -0.31,
    rangeMax =  0.31,
    startValue = 0,)
debug_yaw_angle = bullet.addUserDebugParameter(
    paramName = "yaw angle (rad)",
    rangeMin = -0.36,
    rangeMax =  0.36,
    startValue = 0,)
debug_height = bullet.addUserDebugParameter(
    paramName = "Delta z",
    rangeMin = -200,
    rangeMax = 60,
    startValue = 0,)

# Buttons to change the observing views
debug_initial_view = bullet.addUserDebugParameter(
    paramName="initial view",
    rangeMin = 1,
    rangeMax = 0,
    startValue = 0)
initial_view = bullet.readUserDebugParameter(debug_initial_view)
debug_front_view = bullet.addUserDebugParameter(
    paramName="front view",
    rangeMin = 1,
    rangeMax = 0,
    startValue = 0)
front_view = bullet.readUserDebugParameter(debug_front_view)
debug_top_view = bullet.addUserDebugParameter(
    paramName="top view",
    rangeMin = 1,
    rangeMax = 0,
    startValue = 0)
top_view = bullet.readUserDebugParameter(debug_top_view)

# Button to reset the robot position
reset = bullet.addUserDebugParameter(
    paramName="Reset Position",
    rangeMin = 1,
    rangeMax = 0,
    startValue = 0)
reset_value  = bullet.readUserDebugParameter(reset)

sleep(1)  # Ugly so FIXME
# The initialisation is asynchronous. Wait one second to ensure that the motors reach their initial position before
↪  reading the position values.
ini_pose = a1.current_pose()
while True:
    a1.pose_control(
        roll_angle  = bullet.readUserDebugParameter(debug_roll_angle),
        pitch_angle = bullet.readUserDebugParameter(debug_pitch_angle),
        yaw_angle   = bullet.readUserDebugParameter(debug_yaw_angle),
        Δz          = bullet.readUserDebugParameter(debug_height),
        reference_pose=ini_pose)

    if bullet.readUserDebugParameter(debug_initial_view) != initial_view:
        # set the camera to be initial view
        bullet.resetDebugVisualizerCamera(
            physicsClientId = physics_server_id,
            cameraTargetPosition = [0, 0, 0.4],
            cameraDistance = 1.5,
            cameraYaw = 40,
            cameraPitch = -15)
        initial_view = bullet.readUserDebugParameter(debug_initial_view)
    if bullet.readUserDebugParameter(debug_front_view) != front_view:
        # set the camera to be front view
        bullet.resetDebugVisualizerCamera(
            physicsClientId = physics_server_id,
            cameraTargetPosition = [0, 0, 0.4],
            cameraDistance = 1.5,
            cameraYaw = 90,
            cameraPitch = 0)
        front_view = bullet.readUserDebugParameter(debug_front_view)
    if bullet.readUserDebugParameter(debug_top_view) != top_view:
        # set the camera to be overlook view
        bullet.resetDebugVisualizerCamera(
            physicsClientId = physics_server_id,
            cameraTargetPosition = [0, 0, 0.4],
            cameraDistance = 1.5,
            cameraYaw = 90,
            cameraPitch = -89)
        top_view = bullet.readUserDebugParameter(debug_top_view)

    if bullet.readUserDebugParameter(reset) != reset_value:
        # reset position
        bullet.resetBasePositionAndOrientation(a1.id, [0, 0, 0.43], [0, 0, 0, 1])
        reset_value = bullet.readUserDebugParameter(reset)
```