

**DESIGN AND CONSTRUCTION OF A 4-LEGGED ROBOT IN  
VARIOUS SURFACE ENVIRONMENT**

**OOI JIAN WEI**



**BACHELOR OF MECHATRONICS ENGINEERING WITH  
HONOURS  
UNIVERSITI TEKNIKAL MALAYSIA MELAKA**

**2019**

**DESIGN AND CONSTRUCTION OF A 4-LEGGED ROBOT IN VARIOUS  
SURFACE ENVIRONMENT**

**OOI JIAN WEI**

**A report submitted  
in partial fulfillment of the requirements for the degree of  
Bachelor of Mechatronics Engineering**



**UNIVERSITI TEKNIKAL MALAYSIA MELAKA**

**2019**

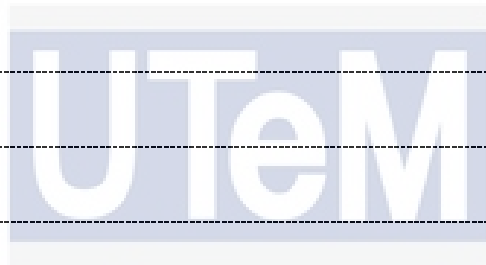
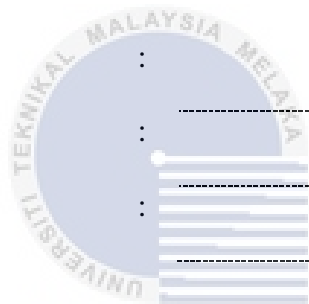
## DECLARATION

I declare that this thesis entitled “DESIGN AND CONSTRUCTION OF A 4-LEGGED ROBOT IN VARIOUS SURFACE ENVIRONMENT is the result of my own research except as cited in the references. The thesis has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature :

Name :

Date :



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

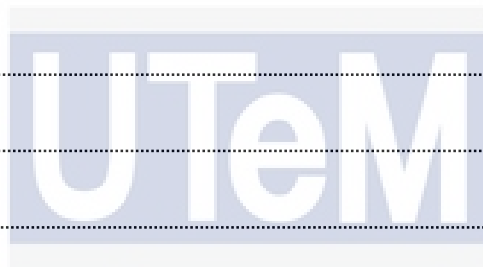
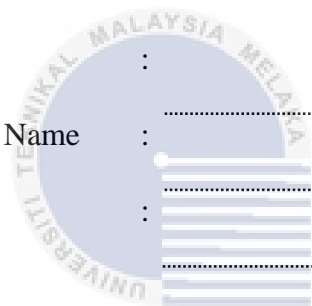
## APPROVAL

I hereby declare that I have checked this report entitled “DESIGN AND CONSTRUCTION OF A 4-LEGGED ROBOT IN VARIOUS SURFACE ENVIRONMENT” and in my opinion, this thesis it complies the partial fulfillment for awarding the award of the degree of Bachelor of Mechatronics Engineering with Honours

Signature :

Supervisor Name :

Date :



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

## DEDICATIONS

To my beloved mother and father

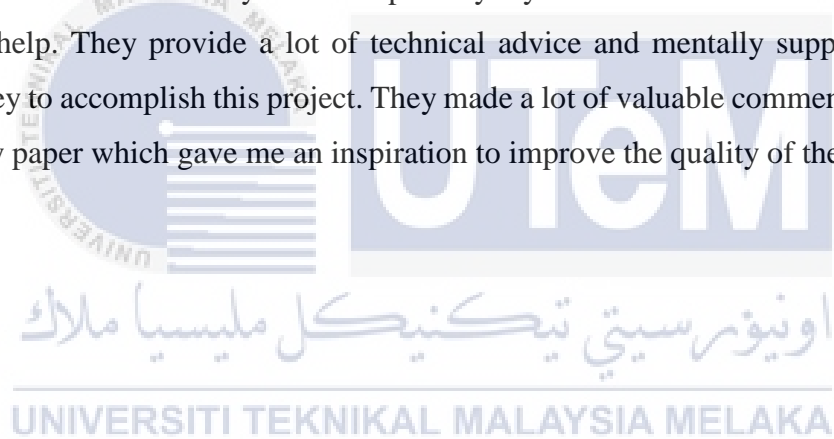


## ACKNOWLEDGEMENTS

In this section, I would like to express my very great appreciation to my dedicate supervisor, Dr Fariz bin Ali @Ibrahim, who was given me support and guidance throughout the project. Without any guidance and support from him, this project would be difficult to complete.

I would like to show my gratitude to my family, especially my parents, Ooi Aik Piew and Lau Yam Keng for their love and support. They provide a lot morale support and encouragement when I was doing this project. I thank them for believing in me and for helping me to keep my spirits high.

I also want to thank to my friends especially my course mates for assisting me when I need help. They provide a lot of technical advice and mentally support during the journey to accomplish this project. They made a lot of valuable comments suggestions on my paper which gave me an inspiration to improve the quality of the project.



## ABSTRACT

This paper presents a study about a 4-legged robot walk in various surface environment with different walking pattern. The main objectives of this paper are designing and constructing a 4-legged walking robot, perform lateral-type and sprawling-type walking pattern on different surface environment. The problems need to solve in this project is to solve the design of the robot that can able to perform both walking pattern. In order to solve this problem, a stable walking algorithm need to be constructed. The first part of this report will be the introduction of the report. In that chapter, the detail of motivation and problem statement will be explained clearly, and the objectives and scope will be stated clearly. Literature review has been done base on previous and latest research which similar to this project. Some of the components had been studied in this part such as the walking pattern of quadruped robot and the degree of freedom (DOF). The third part of this paper will be the methodology which discuss the method used to solve the problem in this project. The software and hardware used in this project will explain in detail. For example, the software used to draw the design of the robot is SolidWorks, and the quadruped robot is programmed by using Arduino IDE. Besides, some of the experiments are carried out to test the accuracy of the robot. Next part of the report will be the result and discussion part. The results from the experiment are discussed in this part to show that the objectives of the research were achieved. The last part of this paper will be the conclusion and recommendation.

## ***ABSTRAK***

Makalah ini membentangkan kajian tentang berjalan kaki robot berkaki empat di pelbagai persekitaran permukaan dengan corak berjalan yang berbeza. Objektif utama makalah ini adalah mereka bentuk dan membina robot berkaki empat, melaksanakan corak jenis dan corak jenis lantang pada persekitaran permukaan yang berlainan. Masalah yang perlu diselesaikan dalam projek ini adalah untuk menyelesaikan reka bentuk robot yang mampu melakukan kedua-dua corak berjalan. Untuk menyelesaikan masalah ini, algoritma berjalan yang stabil perlu dibina. Bahagian pertama laporan ini adalah pengenalan laporan. Dalam bab itu, perincian motivasi dan pernyataan masalah akan dijelaskan dengan jelas, dan objektif dan skop akan dinyatakan dengan jelas. Kajian literatur telah dilakukan berdasarkan penyelidikan sebelumnya dan terkini yang serupa dengan projek ini. Beberapa komponen telah dipelajari di bahagian ini seperti pola berjalan robot empat hulu dan tahap kebebasan (DOF). Bahagian ketiga makalah ini akan menjadi metodologi yang membincangkan kaedah yang digunakan untuk menyelesaikan masalah dalam projek ini. Perisian dan perkakasan yang digunakan dalam projek ini akan menerangkan secara terperinci. Sebagai contoh, perisian yang digunakan untuk menarik reka bentuk robot adalah SolidWorks, dan robot empat kali diprogramkan menggunakan Arduino IDE. Selain itu, beberapa eksperimen dijalankan untuk menguji ketepatan robot. Bahagian seterusnya laporan akan menjadi hasil dan perbincangan. Hasil daripada eksperimen dibincangkan di bahagian ini untuk menunjukkan bahawa objektif penyelidikan telah dicapai. Bahagian terakhir dari kertas ini akan menjadi kesimpulan dan cadangan.



## TABLE OF CONTENTS

	<b>PAGE</b>
<b>DECLARATION</b>	
<b>APPROVAL</b>	
<b>DEDICATIONS</b>	
<b>ACKNOWLEDGEMENTS</b>	<b>2</b>
<b>ABSTRACT</b>	<b>3</b>
<b>ABSTRAK</b>	<b>4</b>
<b>TABLE OF CONTENTS</b>	<b>5</b>
<b>LIST OF TABLES</b>	<b>7</b>
<b>LIST OF FIGURES</b>	<b>9</b>
<b>LIST OF APPENDICES</b>	<b>11</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>12</b>
1.1 Motivation	12
1.2 Problem Statement	13
1.3 Objectives	14
1.4 Scopes	14
<b>CHAPTER 2 LITERATURE REVIEW</b>	<b>15</b>
2.1 Overview of the System	15
2.2 Degree of Freedom	25
2.3 Walking Pattern	26
2.4 Actuator Drive System	30
<b>CHAPTER 3 METHODOLOGY</b>	<b>33</b>
3.1 Project Research	33
3.2 Design of the Structure and Mechanism	35
3.2.1 Design of Qudruped robot	35
3.2.2 Design Consideration	36
3.3 Preview Electronic System and Devices for Quadruped Robot	36
3.3.1 Arduino Mega and Arduino Nano	37
3.3.2 Servo Motor	38
3.3.3 Power Supply for Quadruped Robot	39
3.3.4 nRF24L01 Transceiver Module	40
3.3.5 PS2 Joystick	41
3.3.6 LCD (Liquid Crystal Display)	43
3.4 Mechanical Design	44
3.4.1 SolidWorks Design	44
3.4.2 Material Selection	46

3.5	Electronic Design	48
3.6	Fabrication	50
3.7	Design of Walking Pattern for Quadruped Robot	51
	3.7.1 Angle Calculation for Each Servo Motor	51
	3.7.2 Design for Sprawling Type and Lateral Walking Pattern	53
3.8	Programing for Quaduped Robot and Wireless Remote Controller	54
3.9	Experiments	57
	3.9.1 Experiment 1:Test the error exists for servo motor	57
	3.9.2 Experiment 2: Test the errors occurs for the movement of Quadruped Robot.	58
	3.9.3 Experiment 3: Time taken for the robot passthrough different surface with different walking pattern	59
<b>CHAPTER 4 RESULTS AND DISCUSSIONS</b>		<b>63</b>
4.1	Introduction	63
4.2	Result for Design and Construction of Quadruped Robot	63
	4.2.1 Conceptual Design	63
	4.2.2 Finalize Design	64
	4.2.3 Calibration for Quadruped Robot	66
	4.2.4 Walking Pattern Result	67
4.3	Experiment Results	68
	4.3.1 Accuracy test for servo motors for each leg of the quadruped robot	68
	4.3.2 Accuracy test for quadruped robot while walking in different walking pattern and different movements.	70
	4.3.3 Speed test for the quadruped robot in different type of walking pattern on different surfaces.	72
<b>CHAPTER 5 CONCLUSION AND RECOMMENDATIONS</b>		<b>74</b>
5.1	Conclusion	74
5.2	Future Works	74
<b>REFERENCES</b>		<b>75</b>
<b>APPENDICES</b>		<b>78</b>

## LIST OF TABLES

Table 3.1 Summary Tasks and Experiments which mapped to Objectives.	34
Table 3.2 Design Consideration of the Quadruped Robot	36
Table 3.3 Technical Specification Arduino Mega 2560	37
Table 3.4 Technical Specification of Power HD-1501MG Servo Motors.	39
Table 3.5 nRF24L01 quick reference data	41
Table 3.6 Function for Each Wire of Connector	42
Table 3.7 Pin Description for LCD	44
Table 3.8 First and Second Design for Femur and Tibia.	46
Table 3.9 The position of leg that needed to be decide for both sprawling and lateral walking pattern.	53
Table 3.10 Data Require in Experiment 1	57
Table 3.11 Data Require in Experiment 2	59
Table 3.12 Data Require in Experiment 3	60
Table 4.1 Walking pattern for 1 cycle for sprawling and lateral walking pattern	67
Table 4.2 Exact angle and error occur on servo motor	68
Table 4.3 The offset distances and angles when quadruped walk through 2 m of different surface environments.	71
Table 4.4 The offset distances and calculated error for quadruped walk through 2 m of different surface environments.	71
Table 4.5 Time taken for sprawling type and lateral type of walking pattern walk on different surfaces for 2 meters.	72

Table 4.6 The speed calculated for sprawling type and lateral type of walking pattern walk on different surfaces for 2 meters.



## LIST OF FIGURES

Figure 2.1 Morphology of four-legged robot.	27
Figure 2.2 Schematic walking pattern for various walk.	28
Figure 2.3 Event sequence for different gaits. The limbs: LF, left forelimb; RF, right forelimb; LH, left hindlimb; RH, right hindlimb. Dark colour indicates that the foot is in contact with the ground.[18]	29
Figure 2.4 The gait is developed as a function of time. Each presented frame is taken at a gait event. Solid circles denote a foot in ground contact. White circles denote the placing event of one leg and dashed circles denote the lifting event.[18]	30
Figure 2.5 Picture of HYQ leg prototype with hydraulic cylinder, without compliant element and foot.[16]	31
Figure 2.6 A mammal-type quadruped robot with electrical actuators.	31
Figure 2.7 A sprawling-type quadruped robot with electrical actuators.	32
Figure 3.1: Methodology of Design and Construction	34
Figure 3.2 Design of Four-Legged Robot in SolidWorks	35
Figure 3.3 Arduino Mega 2560 and Arduino Nano	37
Figure 3.4 Structure of Power HD-1501MG Servo Motor	38
Figure 3.5 (i) Buck converter, (ii) T15 Power Plus Battery Charger (iii) 2S1P Lithium ion battery	40
Figure 3.6 nRF24L01 Antenna Wireless Transceiver Module.	41
Figure 3.7 PS2 Joystick and Information of Connector	42
Figure 3.8 LCD 16x2 used in this project	43
Figure 3.9 First and Second Design of Robot's Body.	45

Figure 3.10 Right and Left Joint of Robot.	45
Figure 3.11 3D Printed Robot's Part using PLA	47
Figure 3.12 Relationship Between All Circuit for Quadruped Robot and Remote Controller.	49
Figure 3.13 Schematic Drawing for The Circuit of Quadruped Robot and Remote Controller by Fritzing Software	49
Figure 3.14 Connection of Perf Board.	50
Figure 3.15 Fabrication Process of quadruped robot.	51
Figure 3.16 Parameters Needed to Calculate Angle of Servo Motor.	52
Figure 3.17 Flow chart for wireless remote controller.	55
Figure 3.18 Flow chart for quadruped robot.	56
Figure 3.19 Experiment 1 Set Up	58
Figure 3.20 Experiment 2 Set Up	59
Figure 3.21 Experiment 3 Set Up	62
Figure 4.1 Design of quadruped robot by using hand drawing.	63
Figure 4.2 Final design of quadruped robot in SolidWorks.	64
Figure 4.3 Fabricated quadruped robot.	65
Figure 4.4 Calibration of quadruped robot's legs.	66
Figure 4.5 Sprawling type and lateral type of walking pattern.	68
Figure 4.6 Graph of error occur (%) against the desire angle.	69
Figure 4.7 The experiment carried out on different surface.	70

## LIST OF APPENDICES

APPENDIX A	GANTT CHART	78
APPENDIX B	CODING FOR QUADRUPED ROBOT.	80



# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

Robot is an actuated mechanism programmable in two or more axes with a degree of autonomy, moving within its environment, to perform intended tasks. Autonomy in this context means the ability to perform intended tasks based on current state and sensing, without human intervention.

Robot can be categorized into 2 main types which are “industrial robot” and “service robot”. A service robot is a robot that perform tasks for humans or equipment excluding industrial automation application. Mobile robot can be defined as an automatic machine that is capable to move around in its environment and is not fixed to one physical location.

A quadruped robot can be categorized as a mobile service robot since it can move around in its environment and help human to complete the task given. A quadruped robot is used to carry the object and deliver it to a target places which human is not able to enter due to its flexibility and mobility. Although quadruped robot always used to compare with wheeled robots, however it is well known that legged robots are superior to wheeled robots due to legged robots have better performance on discontinuous terrain than that of wheeled robots and thus quadruped robot will be focus in this research.

This project is motivated by some motivation to start up the research of quadruped robot. The first motivation is from the Sasago Tunnel collapse accidents happen in japan in 2012. This accident caused nine people died and two were injured, making it the deadliest Japanese roadway accident in history[1]. Besides the disaster such as earthquake also cause a lot of people died and injured. If a quadruped robot is well developed, it can help the rescue team go inside the disaster area to search for the survivors.

The second motivation is come from the quadruped robot designed by Bostan Dynamics. It is a high technology research and development company that develop



quadruped robot. The most famous product from this company is the robot called “BigDog”[2]. BigDog was funded by the Défense Advanced Research Projects Agency (DARPA) in the hopes that it will be able to serve as a robotic pack mule to accompany soldiers in terrain too rough for conventional vehicles. Besides, these machines only need a discrete number of isolated footholds, their mobility in unstructured environments can be much higher than their wheeled counterparts, which require a more or less continuous path of support.

Through these motivations, the research about quadruped robot is done in this project and through this research, it can be helpful in the development in quadruped robot in future.

## 1.2 Problem Statement

In this research project, there are few knowledges are required to study which are robotics, programming, electric circuits and the quadruped motion. These knowledges can help to solve the problems that will faced during the research.

In robotics, the knowledge of kinematic is studied, which is a branch of classical mechanics that describes the motion of points, bodies (objects), and systems of bodies (groups of objects) without considering the forces that caused the motion. For the electric circuit knowledge, the connection between the microprocessor and the other electric components are learnt. Programming knowledge help to design the program needed and import it to the microprocessor. For quadruped motion, it is helps to study the terrestrial locomotion in animals using four limbs or legs.

The main problem that need to solve in this project is to decides the motion for the quadruped robot. There are few types of gaits in quadruped motion which are mammal-type and sprawling-type[3]. A mammal-type means the robot which locates its foot vertically downward from the base of the leg as a standard posture. A sprawling-type means the robot whose first leg segment (thigh) is in horizontal direction and second leg segment (shank) is in vertical direction as a standard posture. For both types of quadruped motion, they have their own pros and cons and it must be understood before decides which motion is suitable for the quadruped robot to move on various surface environment.

The next problem is to solve the design of the robot. After decides the motion of the robot, the next step is to design the shape and size of the robot, so it can

perform the motion chosen. The design of the robot must concern about the center of the gravity and also the kinematic of the robot so that the robot can be balance and move.

The other problems are the electric components that required to use such as motors, microprocessor. The microprocessor must decide first so that the other components can fit the microprocessor. After programmed the microprocessor and connected to other electric components, the power must be calculated so that it can use as a reference to choose the required battery capacity and others specification.

### 1.3 Objectives

The objectives of this project are:

- i. To design and construct a 4-legged walking robot controlled by using remote controller.
- ii. To design various movements for the robot to move on flat, tarred and grass surface environments.
- iii. To analyse the walking pattern of the robot.

### 1.4 Scopes

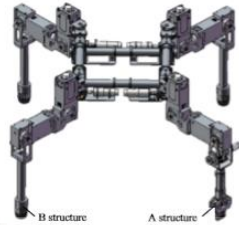
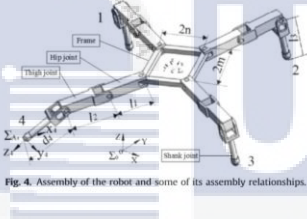
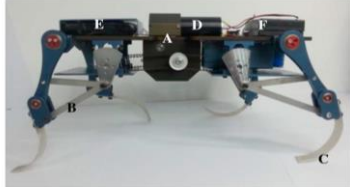
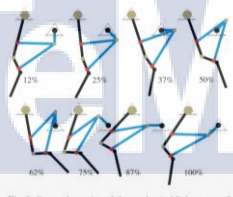
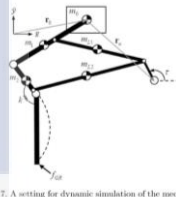
- i. Use 3 degree of freedom for each leg.
- ii. Walking on flat, tarred and grass surface environment.
- iii. Control the movement by using a controller.
- iv. It should perform 2 type of walking pattern.
- v. Using Arduino Mega as the microprocessor
- vi. Use 12 servo motors as actuators
- vii. Can perform forward movement on both walking pattern
- viii. Can perform backward movement on both walking pattern
- ix. Can perform rotational movement on sprawling-type walking pattern

## CHAPTER 2

### LITERATURE REVIEW

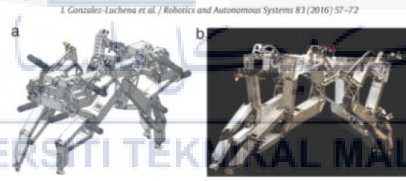
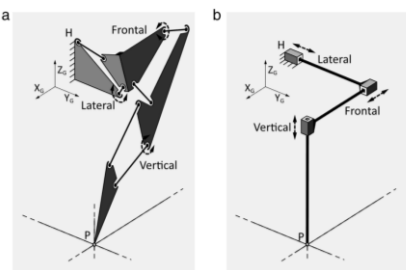
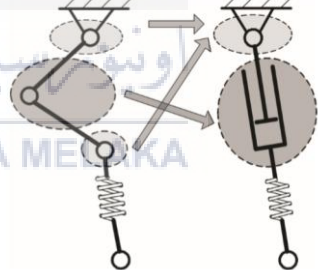
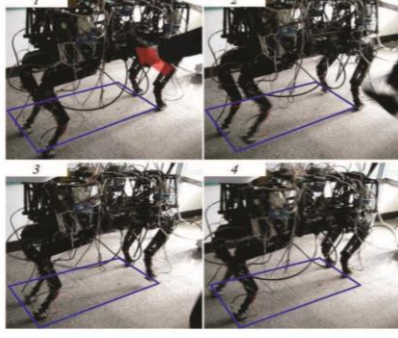
#### 2.1 Overview of the System

Category	Studied articles	
	Journal 1 [4]	Journal 2 [5]
Title	A geometric approach to solving the stable workspace of a hand-foot-integrated quadruped walking robot.	Design of one degree-of-freedom quadruped robot based on mechanical link system: Cheetaroid-II
Aim of this paper	Solving the robot's workspace, including searching steady condition of operation of the robot.	A five-bar linkage mechanism is proposed to emulate the locomotive motion of a leg of a quadruped robot with the reduced number of actuators. (to reduce the weight of robot)
Method	Geometric method of kinematic analysis	Experimental method by observing the locomotion of a dog
Software used	MATLAB, VB and SolidWorks software programs.	MATLAB, Fmincon.m
Walking pattern	Sprawling type	Mammal type
Degree of freedom (each leg)	3	1
Actuator drive system	Servo motor	a Brushless DC (BLDC) motor, a gearing system
Structure and mechanism	Parallel and serial mechanism	five-bar mechanical link system

Dimension of Robot	Length of hip: 67.5mm Length of thigh: 232mm Length of shank: 359mm Length of body: 276mm Width of body: 325mm Mass of body: 8681g Mass of thigh: 1471g Mass of working arm: 1359g	N/A
Design of the robot	 	 <p data-bbox="949 869 1369 958">Fig. 1. A picture of the proposed quadruped robot, Cheetaroid-II. It consists of: <b>A</b>: a main frame, <b>B</b>: five-bar link systems, <b>C</b>: compliant feet, <b>D</b>: a direct-current motor, <b>E</b>: a battery, and <b>F</b>: an embedded controller.</p>  

Category	Studied articles	
	Journal 3 [6]	Journal 4 [7]
Title	A new algorithm to maintain lateral stabilization during the running gait of quadruped robot	Control of a quadruped robot with bionic springy legs in trotting gait
Aim of this paper	i. Calculating the lateral position and speed of the fore swinging leg when it next makes contact with the ground.	i. This paper addresses the problem of trotting control of a quadruped robot with bionic springy legs. The goals are to enhance the robustness of trotting of a quadruped robot and to see

	<p>ii. Controlling the roll angle by mean of inertia forces using the stance leg.</p>	<p>if the quadruped running could be smoother and more stable by introducing certain biological characteristics.</p> <p>ii. The robot reaches the highest speed of <math>2.0 \text{ m}\cdot\text{s}^{-1}</math> and keeps balance under <math>250 \text{ Kg}\cdot\text{m}\cdot\text{s}^{-1}</math> lateral disturbance in the simulations.</p> <p>iii. The effectiveness of these approaches is also verified on a prototype robot which runs to <math>0.83 \text{ m}\cdot\text{s}^{-1}</math> on the treadmill.</p>
Method	Kinetic Momentum Management Algorithm (KMMA)	Robot kinematic analysis
Software used	ADAMS and MATLAB	co-simulation of ADAMS and Matlab/Simulink.
Walking pattern	Mammal type (running gait)	Mammal type
Degree of freedom (each leg)	3	6 Degrees of Freedom (DOF) on torso and 5 DOF on each leg.
Actuator drive system	Brushed DC motors	<p>i. Each leg has four active joints driven by four identical hydraulic cylinders, as well as a passive prismatic spring.</p> <p>ii. The robot has 41 sensors including displacement</p>

		sensors and load cells on the hydraulic cylinders and springs, 3-component force sensors in feet and an Inertial Measurement Units (IMU) on torso.
Structure and mechanism	Uncoupled leg mechanism (three sequentially arranged four-bar mechanism)	N/A
Dimension of Robot	Mass: 43 kg Length: 0.68 m Width: 0.9 m	Mass: 67 kg Length: 0.63 m Width: 0.3 m Height: 0.85 m Thigh length: 0.233 m Shank length: 0.31 m Foot length with spring in normal position: 0.31 m
Design of the robot	 <p><i>I. Gonzalez-Luchena et al. / Robotics and Autonomous Systems 83 (2016) 57-72</i></p> <p>Fig. 3. (a) CAD robot model. (b) Real robot prototype.</p>  <p><i>I. Gonzalez-Luchena et al. / Robotics and Autonomous Systems 83 (2016) 57-72</i></p>	 <p>Fig. 6 Function separation in JCA I.</p>  <p>Fig. 16 Sequence of the marking time trot under lateral disturbance.</p>

Category	Studied articles	
	Journal 5 [3]	Journal 6 [8]
Title	TITAN-XIII: sprawling-type quadruped robot with ability of fast and energy-efficient walking	Kinematic analysis for trajectory generation in one leg of a hexapod robot.
Aim of this paper	Development of a sprawling-type quadruped robot named TITAN-XIII which is capable of high speed and energy efficient walking	Kinematic analysis of a single leg of a hexapod robot is introduced and the trajectory generation is implemented.
Method	<ul style="list-style-type: none"> <li>i. Calculate cost of transport (COT) to get the energy efficiency</li> <li>ii. Forward kinematic</li> </ul>	kinematic and dynamic
Software used	N/A	VRML 2.0, open GL
Walking pattern	Sprawling type	3-types: <ul style="list-style-type: none"> <li>i. Front disposal (Reptilian type)</li> <li>ii. Sagittal disposal (Mammal type)</li> <li>iii. Circular disposal</li> </ul>
Advantage / Disadvantage of walking pattern	<u>Mammal-type</u> <ul style="list-style-type: none"> <li>i. Walk faster than a sprawling-type quadruped robot by utilizing two actuators (e.g., hip and knee) in each leg.</li> <li>ii. Required small torque on each joint by</li> </ul>	N/A

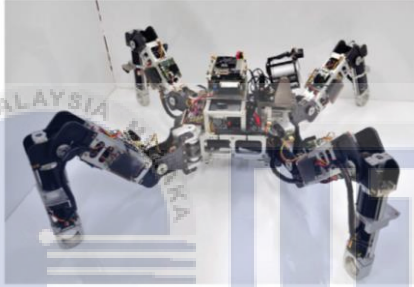
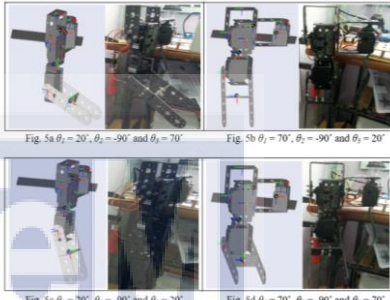
	<p>straightening its leg, especially when the robot stands.</p> <p>iii. Because of the its small footprint, the robot can walk through narrow space or side of a cliff like a mountain goat doing.</p> <p><u>Sprawling-type</u></p> <p>i. High stability, because the robot can locate its centre of gravity at low position and have a wider supporting leg polygon.</p> <p>ii. Wide range of motion because of its proximal yaw axis</p> <p>iii. Since its centre of gravity is low, even if the robot falls down, the damage to the robot is considered to be relatively small.</p> <p>iv. The sprawling first segment, the proximal pitch axis always has to generate the torque to support its own weight, therefore its energy</p>	
--	--	--



	<p>efficiency seems to be low.</p> <p>v. The walking velocity is generated by only proximal yaw actuator, its walking speed would be limited compared to mammal-type.</p>	
Degree of freedom (each leg)	The leg unit has 3DOF and consists of a planar mechanism with 2 DOF employing two pitch axes (Axis 2 and Axis 3), and a yaw axis (Axis 1) which rotates the planar mechanism	3 DOF per leg
Actuator drive system and other components	<p>I. Customized DC brushless motors (FX1206-11 made by Nippo Denki Co., Ltd., max. power 68 W) is used for all of the joint.</p> <p>II. LiFePO<sub>4</sub> battery (26.4 V, 1100 mAh) made by A123</p>	N/A
Structure and mechanism	Wire driven mechanism	N/A
Dimension of Robot	<p>Size (L × W × H): 213.4 × 558.4 × 340.0 mm</p> <p>Weight (w/o battery): 5.29 kg</p> <p>Weight (with battery): 5.65 kg</p> <p>Payload: 5.0 kg</p> <p>Battery: LiFe 26.4V 1100mAh</p>	N/A


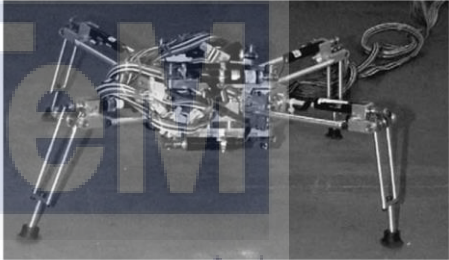
	Battery run time: approx. 20 min.	
Design of the robot	<p>Fig. 3 Technical figure of TDRW-K01 in standard posture</p> <p>Fig. 4 The structure of the leg unit and range of motion of each joint. The detailed drawing of the leg unit and indicates the position of the motor and the encoder.</p>	<p>Fig. 10 Trajectory generation of a set <math>\theta_1, \theta_2, \theta_3</math>. (a) Simulation based on OpenGL.</p>

Category	Studied articles	
	Journal 7 [9]	Journal 8 [10]
Title	Improving traversability of quadruped walking robots using body movement in 3D rough terrains	Modelling and Simulation for One Leg of Quadruped Robot Trajectory
Aim of this paper	Improving the traversability of a quadruped walking robot in 3D rough terrains	Develop a 3 DOF leg which will be used in quadruped robot.
Method	N/A	Forward kinematic
Software used	N/A	SolidWorks 2013 x64 Edition, MATLAB, Arduino, MIT AI2 Companion software (develop android software)
Walking pattern	Sprawling type	N/A
Degree of freedom (each leg)	3 DOF for each leg	3 DOF for each leg

Actuator drive system and other components	N/A	Servomotors with 7kg.cm torque at 6 V rating., Arduino Pro Mini 328 - 5V/16MHz
Structure and mechanism	N/A	N/A
Dimension of Robot	Length of leg: 550mm Length of thigh: 226mm Length of tibia: 226mm Length of body: 226mm	N/A
Design of the robot	 Fig. 16. MRWALLSPECT IV robot.	 Fig. 5a $\theta_1 = 20^\circ$ , $\theta_2 = -90^\circ$ and $\theta_3 = 70^\circ$ Fig. 5b $\theta_1 = 70^\circ$ , $\theta_2 = -90^\circ$ and $\theta_3 = 20^\circ$ Fig. 5c $\theta_1 = 20^\circ$ , $\theta_2 = -90^\circ$ and $\theta_3 = 20^\circ$ Fig. 5d $\theta_1 = 70^\circ$ , $\theta_2 = -90^\circ$ and $\theta_3 = 70^\circ$

Category	Studied articles	
	Journal 9 [11]	Journal 10 [12]
Title	Load distribution and body angle measurement in static walking of quadruped walking robot.	Mechanical design of multifunctional quadruped
Aim of this paper	To obtain some information on load distribution and inclination angle of vehicle body in three typical motion of forward, lateral and turning walk.	Describes the mechanical design of a multifunctional four-legged walking machine that is being developed at the Robotics Research Centre, NTU.
Method	N/A	N/A
Software used	N/A	N/A

Walking pattern	<p>2 types:</p> <ol style="list-style-type: none"> <li>i. Static walking (always keeps the projected centre-of-gravity (CG) point in the support polygon domain.)</li> <li>ii. Dynamic walking (legs are activated fast without falling into statically unstable condition.)</li> </ol>	<p>Many different gaits, such as:</p> <ol style="list-style-type: none"> <li>i. Insect configuration</li> <li>ii. Typical reptile configuration</li> <li>iii. Reptile configuration</li> <li>iv. Intermediate configuration: insect-reptile type</li> <li>v. Intermediate configuration: reptile-mammal type</li> </ol>
Degree of freedom (each leg)	3	3
Actuator drive system and other components	<p>Actuators: DC motors</p> <p>Sensor used:</p> <ol style="list-style-type: none"> <li>i. Ring type load cell for load distribution measurement</li> <li>ii. Inclination sensor for pitch and roll angle measurement (SL-OIJS by Ohmic Co.)</li> </ol>	<p>Servo motors:</p> <p>Hip motor: 4.5 W (MAXON motor) coupled through a 2 stage 19:1 planetary gear box and a 40:1 worm gear.</p> <p>Knee motor: 3 W with a 3 stage 76:1 planetary gearbox and a 40:1 worm gear.</p> <p>All motors are coupled to a 16 counts per turn 2 phase TTL compatible magnet digital encoder.</p>
Structure and mechanism	N/A	<ol style="list-style-type: none"> <li>i. Reptile type</li> <li>ii. Mammalian type</li> <li>iii. “V” Shape configuration</li> <li>iv. Picking object</li> </ol>

		v. LAVA “back to belly” transfer
Dimension of Robot	Length: 0.86m Width: 0.66m Height: 0.92m Number of legs: 4 Number of DOF per leg: 3 Number of DC Motors: 12 $W_L$ : 51N $W_V$ : 314.9N Note: $W_L$ = Leg Weight; $W_V$ = Vehicle Weight	Lengths of machine leg segments are: <ol style="list-style-type: none"> <li>i. Thigh segment: 0.12 m</li> <li>ii. Shank segment: 0.14 m</li> </ol> Body (square): 0.08m Body weight: 1.5-2.0 kg
Design of the robot		

## 2.2 Degree of Freedom

The degree of freedom (DOF) of a mechanical system is the number of parameters that define its configuration. It is defined as the number of independent movements it has and it is important to analyse the system of body in mechanical engineering, aeronautical engineering, robotics, and structural engineering. By knowing the degree of freedom, the exact constraint mechanical design method able to apply in the product design. The exact constraint mechanical design method manages the degrees of freedom to neither under constrain nor over constrain a device. Always begin a design thinking exact constraint. Additional constraints can often cost

more so have a good reason for and fully understand the consequences of over constraint. [13]

Most of the quadruped robots have 1 to 3 degree of freedom for each leg. The designers for the quadruped robot have their own reason when decide for the degree of freedom of the legs for the robot. For example, a quadruped robot called “Cheetaroid-II” was designed in one degree of freedom based on a mechanical link system. The reason to reduce the degree of freedom is reduce the weight if the robot so that it can increase the speed and reduce the energy used of the robot. [5] A quadruped robot with 3 degree of freedom will provide more controllability and thus increase the reachable area of the robot. Therefore, most of the quadruped robots will have 3 degree of freedom for each leg. Some of the designers will limit the minimal number of degrees of freedom, to keep the complexity low and thus increase the robustness, modularity, and ease of maintenance of the system. [14]

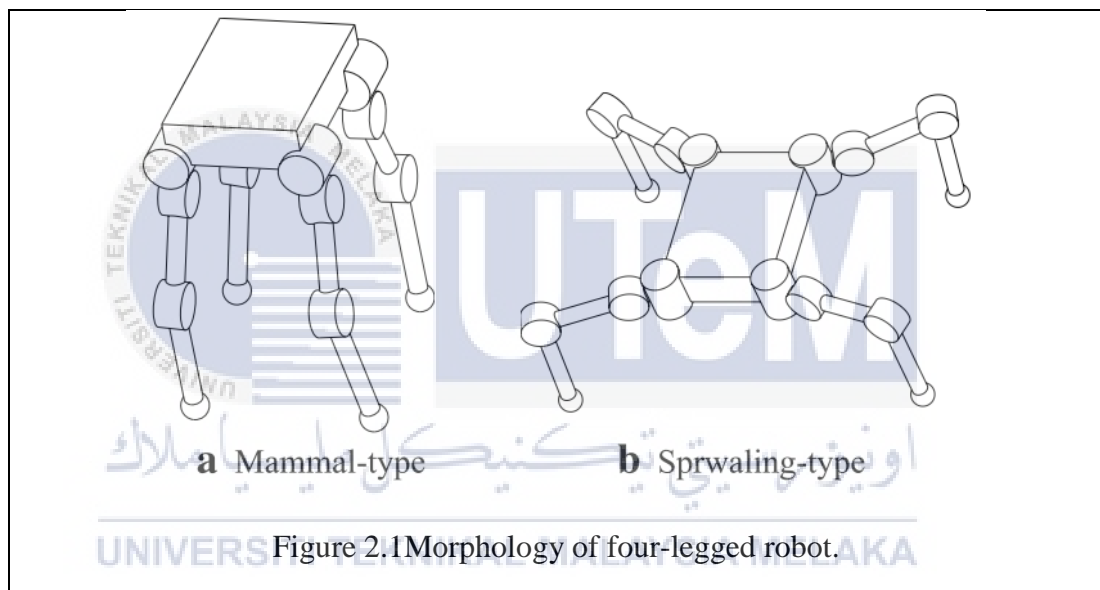
In this project, the quadruped robot will design in 3 degree of freedom for each leg, which means the robot will has 12 degree of freedom in total. There will be 2 DOF in the hip section and the other 1 of the DOF will located in the knee section. The hip section of the quadruped robot will control the leg to swing and lift and the knee section of the quadruped robot will control the location of the end of the leg. Since the quadruped robot in this project is required to perform various movements and move on various surface environments, hence 3 DOF for each leg is the minimum requirement for the robot. If DOF of the quadruped robot is less than 3, the performance of the movement of quadruped robot will be limited.

### **2.3 Walking Pattern**

Recently there are a lot of discussion about a wheeled vehicle and quadruped walking robot. Locomotion on the natural rough terrains where a biped or a wheeled vehicle cannot cope with is one of the most attractive issues for quadruped walking robots[9]. One of the advantage for quadruped robot is the ability to reduce the mechanical coupling between the payload and the terrain thereby enabling irregular terrain traversal[15]. Since these machines only need a discrete number of isolated footholds, their mobility in unstructured environments can be much higher than their wheeled counterparts, which require a more or less continuous path of support[16][17].

Another reason that make quadruped walking robot able to walk on uneven terrain is the walking pattern of the robot that enable it to cope with the uneven terrain.

There are a lot of walking pattern or known as gait for a quadruped robot and most of the walking pattern are adapted from the animal or insect. The walking pattern of the quadruped robot can mainly be classified into 2 main group, a mammal-type and a sprawling-type, according to its leg configuration. A mammal-type means the robot which locates its foot vertically downward from the base of the leg as a standard posture (Figure 2.1(a)). A sprawling-type means the robot whose first leg segment (thigh) is in horizontal direction and second leg segment (shank) is in vertical direction as a standard posture (Figure 2.1(b)).[3]



Mammal-type quadruped robot has several advantages. First, a mammal-type quadruped robot can walk faster than a sprawling-type quadruped robot by utilizing two actuators (e.g., hip and knee) in each leg. Second, a mammal-type quadruped robot required small torque on each joint by straightening its leg, especially when the robot stands. Third, because of the its small footprint, the robot can walk through narrow space or side of a cliff like a mountain goat doing [3].

On the other hand, a sprawling-type (reptile-type) quadruped robot has its features. First, the stability of the robot is high, because the centre of gravity of the robot can be located at low position and have a wider supporting leg polygon. Second, the robot has wide range of motion because of its proximal yaw axis, therefore it can choose foot placement widely. Third, since its centre of gravity is low, the damage to

the robot is considered relatively small when the robot falls down. Additionally, it is easy to use the body of the robot as a fifth foot depending on the terrain[3].

For a sprawling-type quadruped robot, it can divide into 2 types of walking, one is static walking which always keeps the projected centre-of-gravity (CG) point in the support polygon domain. The support polygon domain is created by the 3 unmoved legs which make a tripod stance. The other is dynamic walking where legs are activated fast without falling into statically unstable condition. For a static walking, it can basically perform 3 basic type of walking, which are forward walk, lateral walk and turning walk. In this project, the walking pattern of sprawling-type in forward walk and lateral walk will be analyzed, since both sprawling-type walking patterns has its own advantages to move on certain terrain.

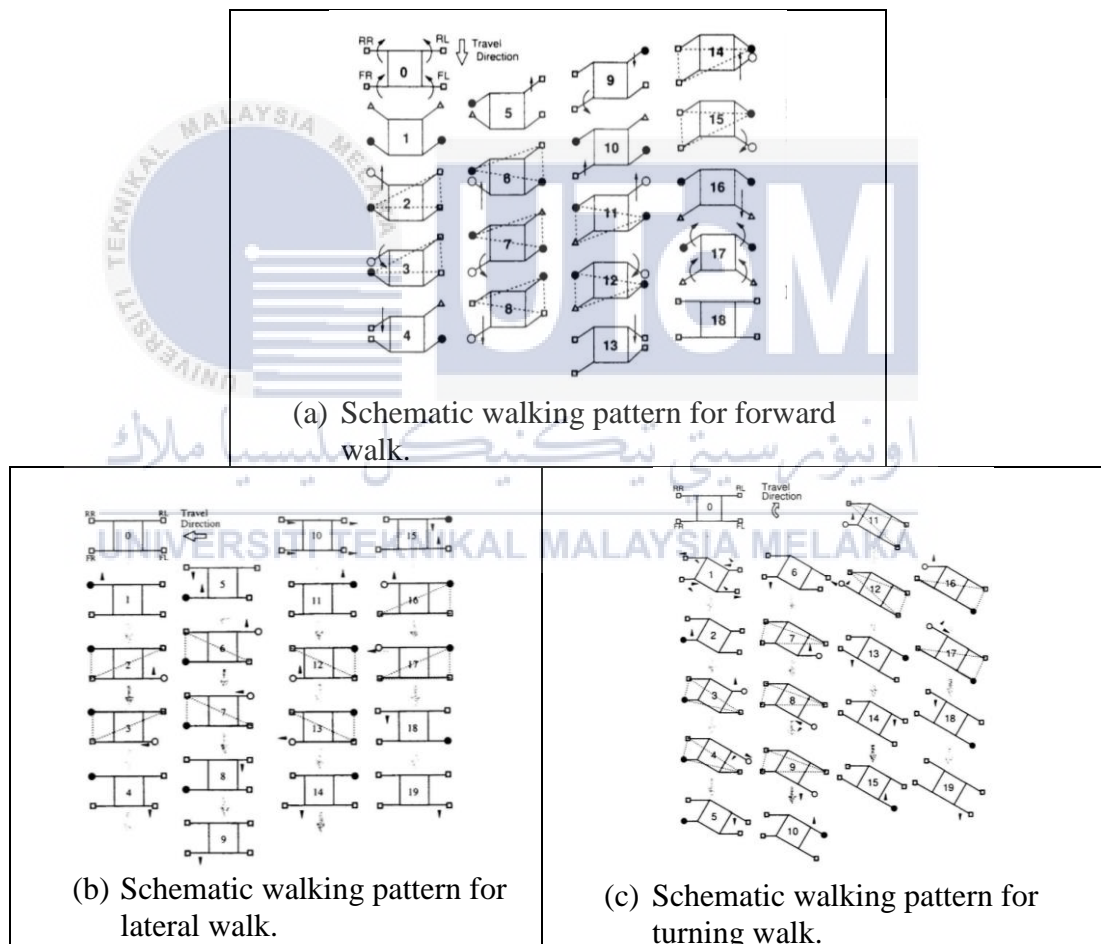


Figure 2.2 Schematic walking pattern for various walk.

As shown in Figure 2.2, lifting of legs are always lift one and move forward at a time. At the same time, the other 3 legs will rotate the joint at hip to move the centre-of-gravity (CG) into the support polygon. It will make the robot stay in a stable position



when moving forward. The method for lateral walk and turning walk are similar with forward walk. The body of the robot will move toward to the support polygon to maintain its stability when lifting one of the legs forward or turning [11].

As mentioned, a mammal-type quadruped robot it can perform a higher speed movement compare to sprawling type quadruped robot. In this walking pattern, robot is able to perform crawl gait and also trot gait. When increasing the speed during the walking of the robot, the robot continuously change from a crawl gait to a trot gait, in order to achieve a smoother quadruped locomotion[18]. In robotics, in order to achieve smooth walking from low speed to high speed, these gaits should be similarly switched continuously. This can be easily achieved by applying the wave gait rule [17,18], i.e., the interlimb phase relationships should follow the value of the duty factor for a changing speed. This rule improves the stability of the locomotion and maximizes the stability margin [17,18] because the support of the body smoothly changes from three-point support (walk) to two-point support (trot) [17–19].

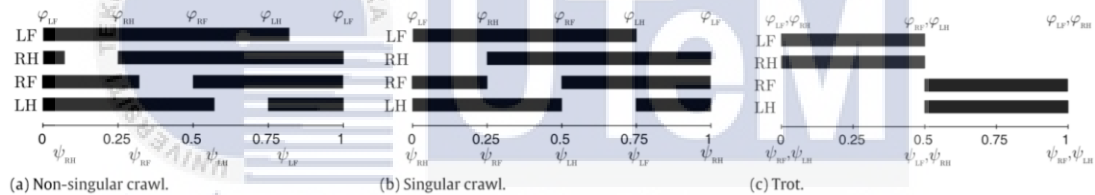
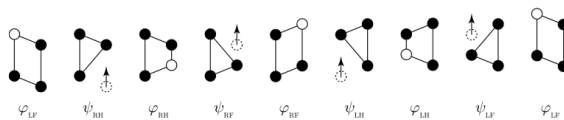
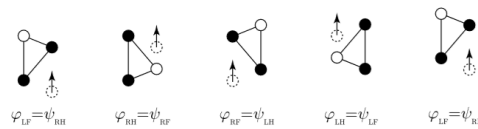


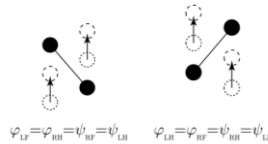
Figure 2.3 Event sequence for different gaits. The limbs; LF, left forelimb; RF, right forelimb; LH, left hindlimb; RH, right hindlimb. Dark colour indicates that the foot is in contact with the ground.[18]



(a) Non-singular crawl gait.



(b) Singular crawl gait



(c) Gait diagram for the trot gait.

Figure 2.4 The gait is developed as a function of time. Each presented frame is taken at a gait event. Solid circles denote a foot in ground contact. White circles denote the placing event of one leg and dashed circles denote the lifting event.[18]

Figure 2.3 and Figure 2.4 shown the event sequence for different gaits and the gait is developed as a function of time respectively. In non-singular crawl gait, only 1 leg will denote the lifting event and continue with denote a foot in ground contact on the same time. After The next step will start when the previous step finish denotes a foot in ground. For a singular crawl gait, when a leg is ready to denote the placing event, the diagonal leg will denote the lifting event. When the leg is contacted to ground, the diagonal leg will denote the placing event and the leg parallel to the placing event leg will denote the lifting event. These events will repeat as a loop. For a trot gait, the diagonal legs will pair up and do a similar motion. When a pair of leg (pair A) contacted to ground, the other pair (pair B) will present a lifting event follow by placing event. After the pair of legs (pair B) touch the ground, the leg of pair A will repeat the same event that pair B presented.

In this project, both mammal-type and sprawling-type will be used. Both types of walking pattern have its own advantages and disadvantages. Therefore, combine two type of walking pattern can increase the ability of the robot to walk on various surface.

## 2.4 Actuator Drive System

An actuator is something that used to converts energy into motion. It also can be used to apply a force. Basically, actuator can be divided in to 3 types, which are hydraulic, pneumatic and electric actuators. In most of the research shows that hydraulic actuators, pneumatic actuators and electrical actuators are most commonly been used. All type of the actuators has its own advantages and disadvantages.

For the most advanced quadruped BigDog [20, 21], developed by Boston Dynamics, is driven by hydraulic actuation and uses force controlled architecture in

combination with passive linear pneumatic compliance in the lower leg to be able to cope with unstructured terrain.[16] One of the advantage of using hydraulic actuation is they have high power to weight ratio and thus are the smallest actuation option available. Besides, hydraulic actuation has a high control bandwidth due to hydraulic fluid is generally incompressible.



Figure 2.5 Picture of HYQ leg prototype with hydraulic cylinder, without compliant element and foot.[16]

As shown in Figure 2.5, it is a HYQ leg prototype with hydraulic cylinder. The design is a mammal skeletons design which show the configuration of the front and hind legs form an x-shape in the sagittal plane. Although hydraulic actuator is good as a actuator of a quadruped robot, but it is more complicated to assemble and design compare to electrical actuators such as servo motors.

There are two type of electrical actuators which commonly used in a quadruped robot, which are dc motor and servo motor. Electrical actuators are commonly used because it is easy to control its speed and position.

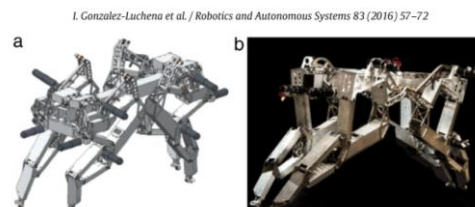


Figure 2.6 A mammal-type quadruped robot with electrical actuators.

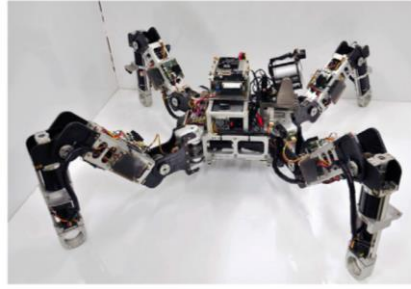


Figure 2.7 A sprawling-type quadruped robot with electrical actuators.

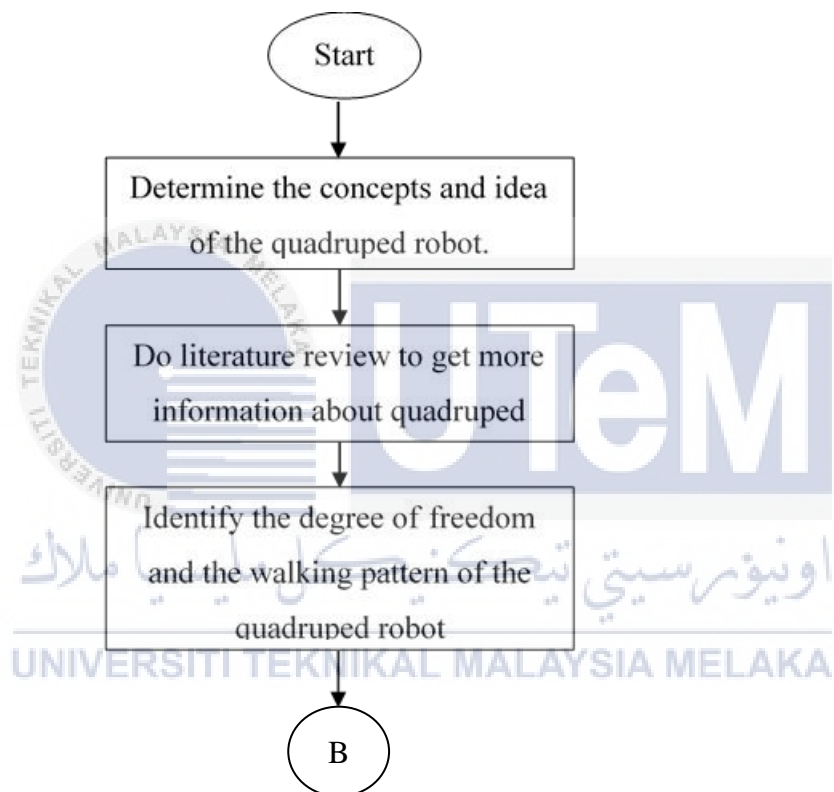
Figure 2.6 and Figure 2.7 shown different type walking pattern quadruped robot with electrical actuators. In this project, electrical actuator is more suitable to use because electrical actuators easy to control and can easy to change the type of walking pattern of the robot. It can change the walking pattern of the robot by changing the turning angle of the motor and it can just change it by programming controller to control the turning angle of the motors.



## CHAPTER 3

### METHODOLOGY

#### 3.1 Project Research



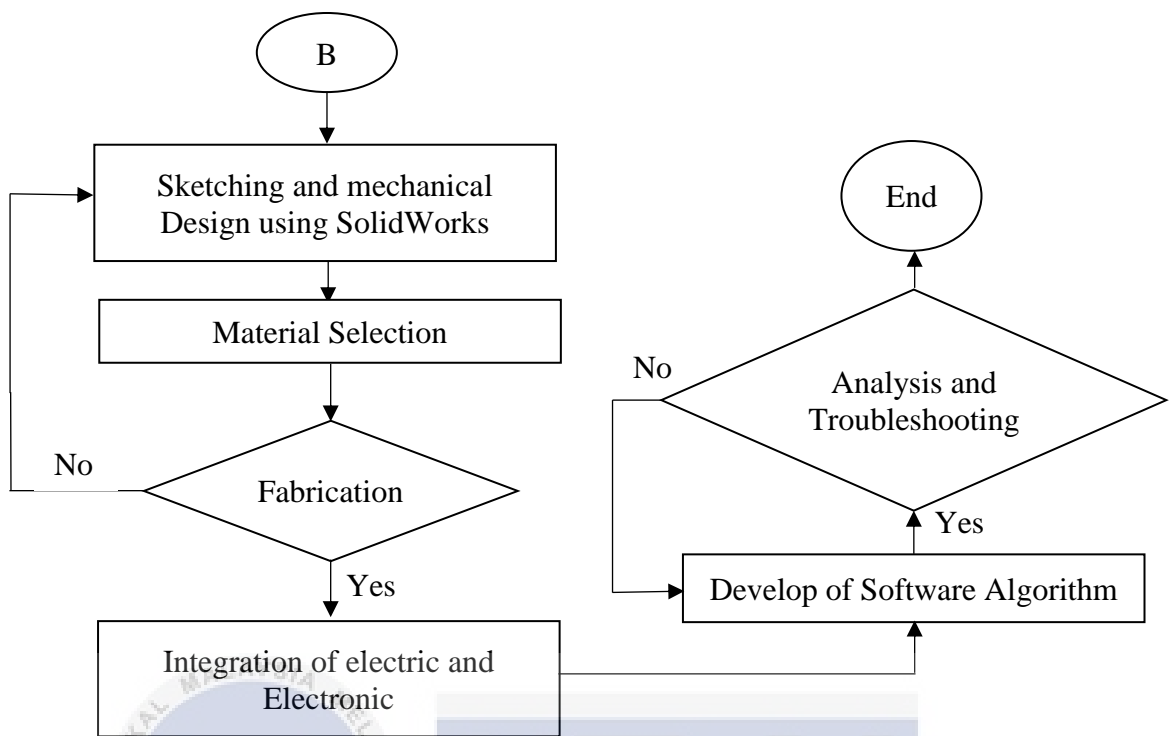


Figure 3.1: Methodology of Design and Construction

The tasks and experiments in this project is mapped to the main objective this project. Table 3.1 shown the experiment that mapped to the objective of this project.

Table 3.1 Summary Tasks and Experiments which mapped to Objectives.

	Objectives		
	1	2	3
Task 1	✓	✓	
Task 2			✓
Experiment 1	✓		
Experiment 2	✓		
Experiment 3			✓

## 3.2 Design of the Structure and Mechanism

The design of the structure and mechanism of the quadruped robot is based on the walking pattern of the robot, degree of freedom of the robot, the materials used for the body of robot and the electronic device used for the robot. These considerations stated will affected by each other when doing the design for the robot. Therefore, the scope listed is used to limit the design of the robot so that it will has a clear goal for the design of the robot.

### 3.2.1 Design of Qudruped robot

The idea of conceptual design for the quadruped robot is comes from some thesis and journals that had been done by previous researcher. In this project, the design of the robot will perform forward and lateral sprawling type walking pattern for the quadruped robot.

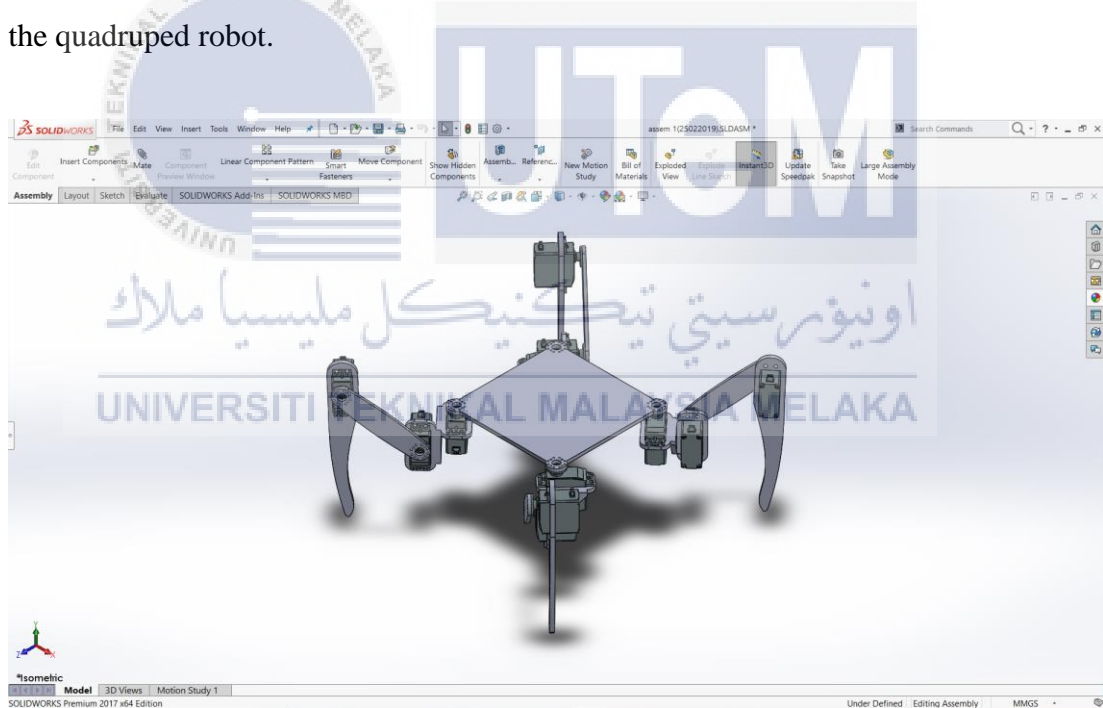


Figure 3.2 Design of Four-Legged Robot in SolidWorks

Figure 3.2 shows the design of four-legged robot in SolidWork. In the design of robot, it is decided to use 12 servo motors as the actuators of robot. That means each leg will be attached by 3 servo motors and produce 3 Degree of Freedom (DOF). For the servo motors, it will mount on joint, femur and tibia of the leg which can help the robot to perform a smooth movement. The size of the body for the robot is 15cm x

15cm and the length of femur and tibia are 10cm and 16cm. The ratio for the length of femur and tibia is based on the golden ratio which is 1.618.

### 3.2.2 Design Consideration

The consideration for the design of the robot will be the walking pattern of the robot, degree of freedom of the robot, the materials used for the body of robot and the electronic device used for the robot. As mentioned before, each of them will affect each other and come out with a different design.

Table 3.2 Design Consideration of the Quadruped Robot

Design considerations	Selected option	Description
Walking pattern	Sprawling-type with forward and lateral walking pattern.	To walk on different surface, both type of walking patterns is used.
Degree of Freedom (DOF) for each leg	3 DOF	3DOF is suitable for both walking pattern of the quadruped robot.
Material used	PLA (Polylactic Acid) for 3D Printing	3D printing is easier to fabricate, and its dimension accuracy will be higher and price will be cheaper compared to other material and method.
Electronic devices	Arduino and servo motors.	Arduino is easy to be programmed and the price is relatively cheap. Servo motors are chosen due to their rotation angle being easy to control.

### 3.3 Preview Electronic System and Devices for Quadruped Robot

In this section, a deep research for the electronic system will be done to understand well for the requirements and constraints of the project. After the research for the electronic system, the microcontroller and other electronic components will be decided and compiled together.



### 3.3.1 Arduino Mega and Arduino Nano

Arduino Mega 2560 is chosen as the microcontroller used in this project. The reason to choose Arduino Mega 2560 as the microcontroller is because it has more I/O port compare to Arduino Uno and Arduino Nano. Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started. he Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328P (Arduino Nano 3.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. Figure 3.3 below shown the microcontroller (Arduino Mega 2560) and Arduino Nano used in this project, and the Table 3.3 shown the technical specification of the Arduino Mega 2560 and Arduino Nano.

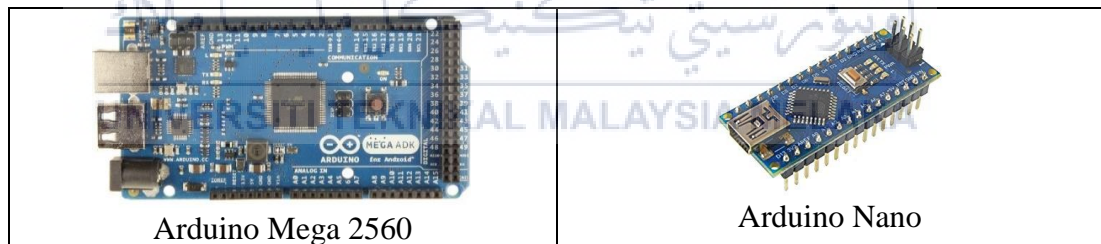


Figure 3.3 Arduino Mega 2560 and Arduino Nano

Table 3.3 Technical Specification Arduino Mega 2560

<b>Microcontroller</b>	ATmega2560	ATmega328
<b>Operating Voltage</b>	5V	5V
<b>Input Voltage (recommended)</b>	7-12V	-
<b>Input Voltage (limit)</b>	6-20V	-
<b>Digital I/O Pins</b>	54 (of which 15 provide PWM output)	22 (6 of which are PWM)
<b>Analog Input Pins</b>	16	8

<b>DC Current per I/O Pin</b>	20 mA	-
<b>DC Current for 3.3V Pin</b>	50 mA	-
<b>Flash Memory</b>	256 KB of which 8 KB used by bootloader	32 KB of which 2 KB used by bootloader
<b>SRAM</b>	8 KB	2 KB
<b>EEPROM</b>	4 KB	1 KB
<b>Clock Speed</b>	16 MHz	16 MHz
<b>LED_BUILTIN</b>	13	-
<b>Length</b>	101.52 mm	45mm
<b>Width</b>	53.3 mm	18mm
<b>Weigh</b>	37 g	7 g

### 3.3.2 Servo Motor

After some research for the actuator, servo motors are chosen to use as the actuator for this project due to easy control and accuracy of the motor. A servomotor is a closed-loop servomechanism that controls its motion and final position by using position feedback. The input to its control is a signal that represents the position commanded for the output shaft, either analog or digital.

In this project, Power HD-1501MG servo motors are used as the actuator for the robot. The advantages of using this serv motor as the actuator is because it has high torque and the accuracy of the servo motors are within  $\pm 10^\circ$ . Figure 3.4 shows the structure of the Power HD-1501MG servo motor and Table 3.4 below is the technical specification of the Power HD-1501MG servo motors.

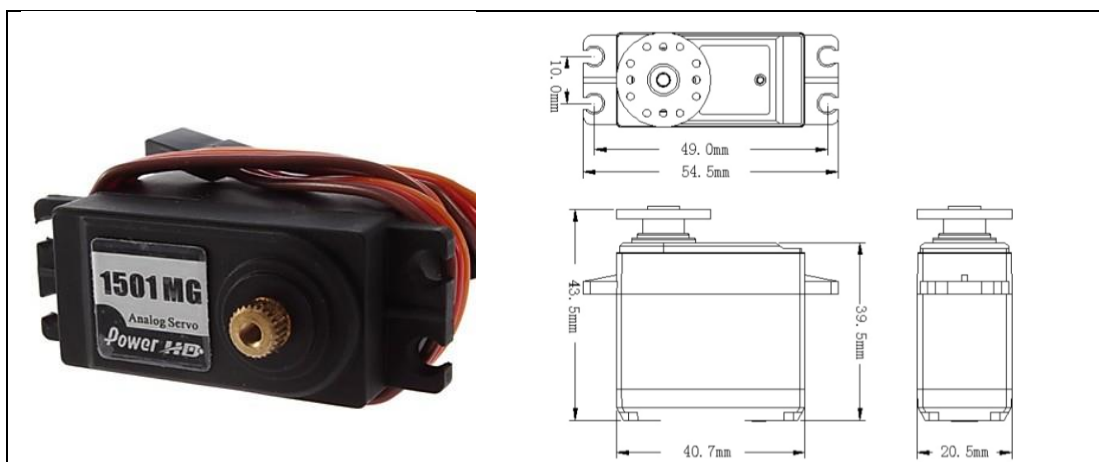


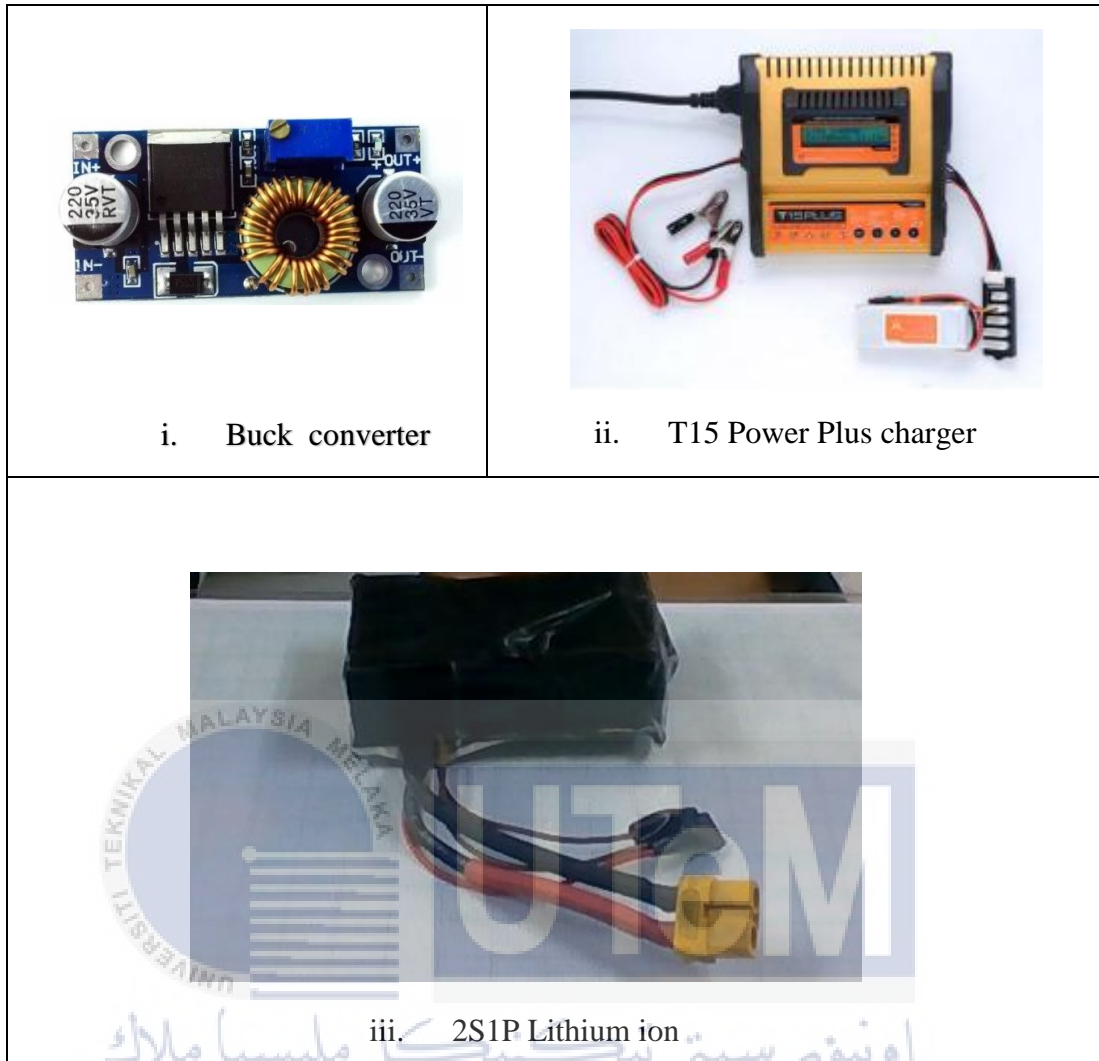
Figure 3.4 Structure of Power HD-1501MG Servo Motor

Table 3.4 Technical Specification of Power HD-1501MG Servo Motors.

Item	Specification
Storage Temperature Range	-20°C~60°C
Operating Temperature Range	-10°C~50°C
Operating Voltage Range	4.8V-6.0V
Operating speed (at no load)	0.16 sec/60°
Running current (at no load)	400mA
Stall torque (at locked)	15.5 kg-cm
Stall current (at locked)	2300 mA
Idle current (at stopped)	4mA
Limit angle	180° ± 10°
Weight	63 ± 1g
Neutral position	1500 μ sec

### 3.3.3 Power Supply for Quadruped Robot

Power supply is very important to the quadruped robot because it is used to power up the microcontroller and also the servo motors. In this project, lithium ion batteries are used to power up all the electronic components for the robot. Lithium ion batteries are chosen because it is rechargeable, it has more capacity and has a higher power output. To prevent a noisy power supply to micro controller, 2 separate lithium ion batteries are used. The lithium ion batteries that are used in this project are connected in 2 Series 1 Parallel (2S1P). The voltage supply will be 7.4V and the capacity will be 3000mAh. The charger used to charge lithium ion batteries is T15 Power Plus charger. T15 Power Plus is an innovative multi-function charger with built-in lithium balancer, designed to maximize charge efficiency using its advanced charging algorithm with precise digital control of the charging and discharging process. Since most of the electronic components are power up by 5V, hence a buck converter is needed to step down the voltage from 7.4V to 5V.



— Figure 3.5 (i) Buck converter, (ii) T15 Power Plus Battery Charger  
(iii) 2S1P Lithium ion battery

### 3.3.4 nRF24L01 Transceiver Module

In this project, the quadruped robot is controlled by a wireless remote control. Hence 2 wireless communication modules are needed to transmit and receive data. nRF24L01 is a single chip radio transceiver for the worldwide 2.4 - 2.5 GHz ISM band. The transceiver consists of a fully integrated frequency synthesizer, a power amplifier, a crystal oscillator, a demodulator, modulator and Enhanced ShockBurst™ protocol engine. Output power, frequency channels, and protocol setup are easily programmable through a SPI interface. Current consumption is very low, only 9.0mA at an output power of -6dBm and 12.3mA in RX mode. Built-in Power Down and Standby modes

makes power saving easily realizable. Figure 3.6 below shows the nRF24L01 chip and Table 3.5 shows the quick reference data for nRF24L01 chip.

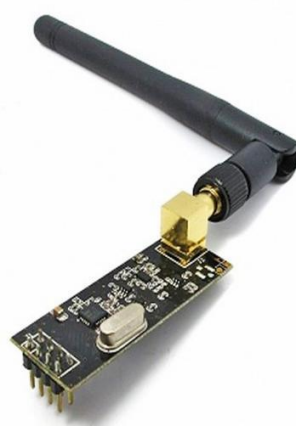


Figure 3.6 nRF24L01 Antenna Wireless Transceiver Module.

Table 3.5 nRF24L01 quick reference data

Parameter	Value	Unit
Minimum supply voltage	1.9	V
Maximum output power	0	dBm
Maximum data rate	2000	Kbps
Supply current in TX mode @ 0dBm output power	11.3	mA
Supply current in RX mode @ 2000 kbps	12.3	mA
Temperature range	-40 to +85	°C
Sensitivity @ 1000 kbps	-85	dBm
Supply current in Power Down mode	900	nA

### 3.3.5 PS2 Joystick

In this project, the quadruped robot is controlled by a wireless remote control. The controller that are used in this project is PS2 joystick controller which connect to the Arduino Nano and transmit the signal through nRF24L01. The reason to choose this remote control is because it has more button to use and less wire used to talk to controller. Figure 3.7 below shows the PS2 joystick and the information of the connector for PS2 joystick and Table 3.6 shows the function for each wire of connector.



Figure 3.7 PS2 Joystick and Information of Connector

Table 3.6 Function for Each Wire of Connector

Brown - Data:	This is an open collector output and requires a pull-up resistor (1 to 10k, maybe more). (A pull-up resistor is needed because the controller can only connect this line to ground; it can't actually put voltage on the line).
Orange - Command:	Controller.
Grey - Vibration Motors Power:	6-9V? With no controller connected, this measures about 7.9V, with a controller, 7.6V, most websites say this is 9V (except playstation.txt -> 7.6V), although it will still drive the motors down around 4V, although somewhat slower. When the motors are first engaged, almost 500mA is drawn on this line, and at steady state full power, ~300mA is drawn.
Black - Ground	
Red - Power:	Many sites label this as 5V, and while this may be true for PlayStation 1 controllers, we found several wireless brands that would only work at 3.3V. Every controller tested worked at 3.3V, and the actual voltage measured on a live PlayStation talking to a controller was 3.4V. McCubbin says that any official Sony controller should work from 3-5V. Most sites say there is a 750mA fuse for both controllers and memory cards, although this may only apply to PS1's since 4 dual shock controllers could exceed that easily.
Yellow - Attention:	This line must be pulled low before each group of bytes is sent / received, and then set high again afterwards. In our testing, it wasn't sufficient to tie this permanently low--it had to be driven down and up around each set. Digitan considers this a "Chip Select" or "Slave Select" line that is used to address different controllers on the same bus.

Blue - Clock:	500kHz, normally high on. The communication appears to be SPI bus. We've gotten it to work from less than 100kHz up through 500kHz (500k bits / second, not counting delays between bytes and packets). When the guitar hero controller is connected, the clock rate is 250kHz, which is also the rate the PlayStation 1 uses.
White - Unknown	
Green - Acknowledge:	This normally high line drops low about 12us after each byte for half a clock cycle, but not after the last bit in a set. This is an open collector output and requires a pull-up resistor (1 to 10k, maybe more). playstation.txt says that the PlayStation will consider the controller missing if the ack signal (> 2us) doesn't come within 100us.

### 3.3.6 LCD (Liquid Crystal Display)

LCD (Liquid Crystal Display) is used in this project to preview the mode chosen by users. Since this robot has 2 walking pattern which are forward and lateral movement in sprawling, hence a display is needed to show the current walking pattern set by user. In this display, it will also show the speed level added to the robot and the height of robot added or minuses by user. The type of LCD used is a basic 16x2 LCD display. A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. Figure 3.8 below shows the LCD used in this project and Table 3.7 shows the Pin description for this LCD.



Figure 3.8 LCD 16x2 used in this project

Table 3.7 Pin Description for LCD

Pin No	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	Vcc
3	Contrast adjustment; through a variable resistor	VEE
4	Selects command register when low; and data register when high	Register Select
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7
15	Backlight VCC (5V)	Led+
16	Backlight Ground (0V)	Led-

### 3.4 Mechanical Design

#### 3.4.1 SolidWorks Design

This section will show the design of body, joint, femur and tibia of quadruped robot. This is a very important part for the project because it will affect the stability of the robot, the coding written for the microprocessor and the size of robot. Hence there are some consideration need to be concern during the design of robot.

The first part to design for the robot is the body of the robot. The things that need to be concern is the size of the robot. The size of robot is decided base on the size of microcontroller used in this project. The size of Arduino Mega that used in this project is 10.2cm x 5.3cm, hence width for the body of robot is decided in 15cm to fit in the microcontroller. In order to simplify the calculation, the body is design in symmetry which is 15cm x 15cm.



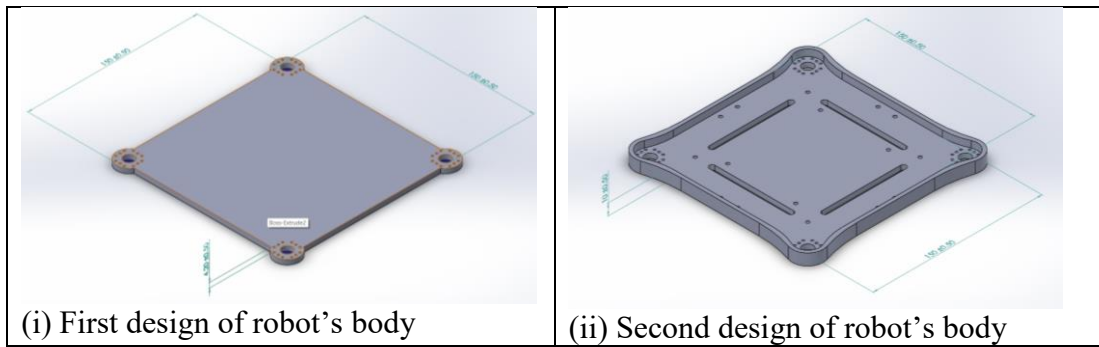


Figure 3.9 First and Second Design of Robot's Body.

Figure 3.9 shown that the first and second design of robot's body. The body of robot is design in square shape due to easy fabricate and simplify the calculation for the walking pattern. The four corners of the body are designed to fit in the horns of servo motor. In the first design, it is less tough and easy to twist. Therefore, second design is designed by add an I-beam on the frame of body to strengthen up the strength of body. This can be understood as a body added a bone. Besides, the body is added some holes to let some wire pass through.

After finishing the design of body, the joint of robot is designed to mount 2 servos in to 1 joint. The thing that need to be concern when design the joint is the size to fit servo motor. The place to mount servo motor should be almost same dimension with servo motor so that it will not be too loose or too tight to fit it. Figure 3.10 below shows the design of joint in solid work.

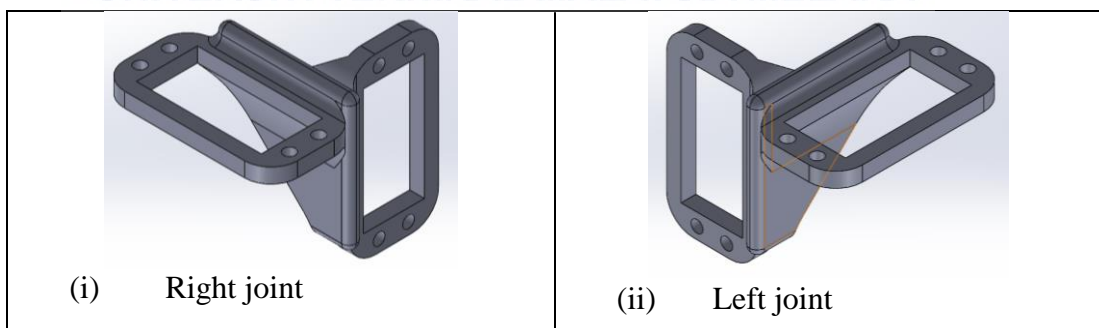

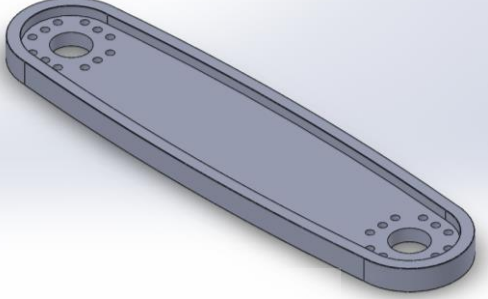




Figure 3.10 Right and Left Joint of Robot.

The next part is the femur of robot. The length of femur is design in 10cm and in the end to end of femur are also has holes to mount servo motor. The first design of femur has the same problem as the body design, it will have twisting problem when during movement. Hence, the second design also added I-beam to strengthen up the

strength of femur. For the length of tibia, it is designed in 16cm. The ratio for tibia and femur is base on golden ratio 1.618. There is also has 2 design for tibia. This is because the first design is too weak to hold the body weight of the quadruped robot. Therefore, the thickness of tibia is increased to strengthen the base. Table 3.8 below show the first and second design for femur and tibia.

Table 3.8 First and Second Design for Femur and Tibia.

Femur Design	 <p>(i) First design</p>	 <p>(ii) Second design</p>
Tibia Design	 <p>(i) First design</p>	 <p>(ii) Second design</p>

### 3.4.2 Material Selection

After finishing the design of quadruped robot, it will come to fabrication process. However, before fabrication, material used to build the robot need to be decided. There are some choices can use to fabricate quadruped robot, such as acrylic, carbon fiber, stainless steel and Polylactic Acid (PLA).

There are some considerations need to be concern before select the material. The first thing to be concern is the prices of material. Since the budget for this project is limited, hence the price of material needs to be reasonable to distribute the money evenly for other components such as electronic components and also to prevent the project overbudget. The next thing to be concern is the strength of the material. The

material used to fabricate the robot need to be able to withstand the weight of the body and tough enough to withstand some impact when the robot falls down. The last thing to concern before fabricating the robot is difficulty of fabrication by using the material. Since this project will has some prototyping to finalize the design of robot, hence a material that is easy to fabricate is very important so that is will not a lot of waste the time to build the robot.

After some researches are done, Polylactic Acid (PLA) is chosen as the material used to fabricate the quadruped robot. The first reason is PLA is relatively cheap compare to acrylic and stainless steel. After the research about the price for materials, the price for acrylic, carbon fiber and stainless steel are few times expensive than PLA and that is the reason PLA is chosen. Beside PLA is able to withstand the weight of the body of robot and tough enough withstand some impact when the robot falls down. Lastly, PLA is easy to fabricate compare to other materials. PLA is the material used for 3D printer and it is easy to fabricate the shape of the robot. As long as the design of the robot come with CAD drawing, then the 3D is able to print out the part that we design. Compare to other material such as acrylic, carbon fiber and stainless steel, they need to use milling machine or lathe machine to fabricate the part of robot, which it will need more effort and time to fabricate it. Therefore, Polylactic Acid (PLA) is chosen as the material used for the robot. Figure 3.11 shows the part of quadruped robot that 3D printed by using material Polylactic Acid (PLA).

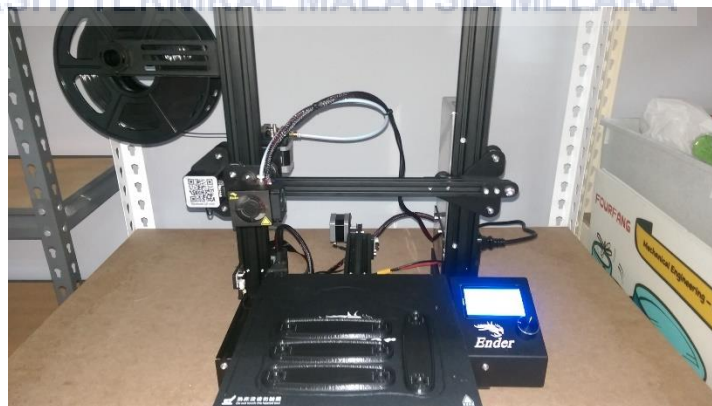


Figure 3.11 3D Printed Robot's Part using PLA

### 3.5 Electronic Design

In this section, the design of circuit for the quadruped robot and remote controller will be shown and explained. The microcontroller used for the quadruped robot will be Arduino Mega 2560. For the actuators, 12 servo motors are used, which means each leg will attach 3 servo motors. The robot will communicate with remote control through a transceiver call nRF24L01. All of these electronic component will connected together to a perf board.

The power supply used for this project will be a 2S1P lithium ion battery, which will provide a 7.4 V for the circuit. However, this voltage is not suitable for the electronic components. Therefore, a buck conveter is used to step down the voltage from 7.4 V to 5.0V. The require voltage needed for servo motor is same as the microcontroller. However servo motor will draw a lot of ampere when the robot is walking, and it will cause a voltage drop from the power supply. The voltage drop will cause microcontroller restart affect the performance of robot. Therefore, separate power supplies are used for microcontroller and servo motors to solve this problem.

Besides quadruped robot, the circuit for remote controller is also important. The micro processor used for remote controller is Arduino Nano. This is because the I/O port require for the remote controller si lesser than the robot. Hence a small microprocessor is enough to use. For the circuit connection, it need a nrRF24L01 transceiver to transmit the data. A PS2 joystick is also connect to Arudino Nano to use as an input and a LCD display is used to display the current mode and information of robot. Figure 3.12 below shows the relationship between all circuit for the robot and remote controller. Figure 3.13 shows the schematic drawing for the circuit of quadruped robot and remote controller by Fritzing Software.

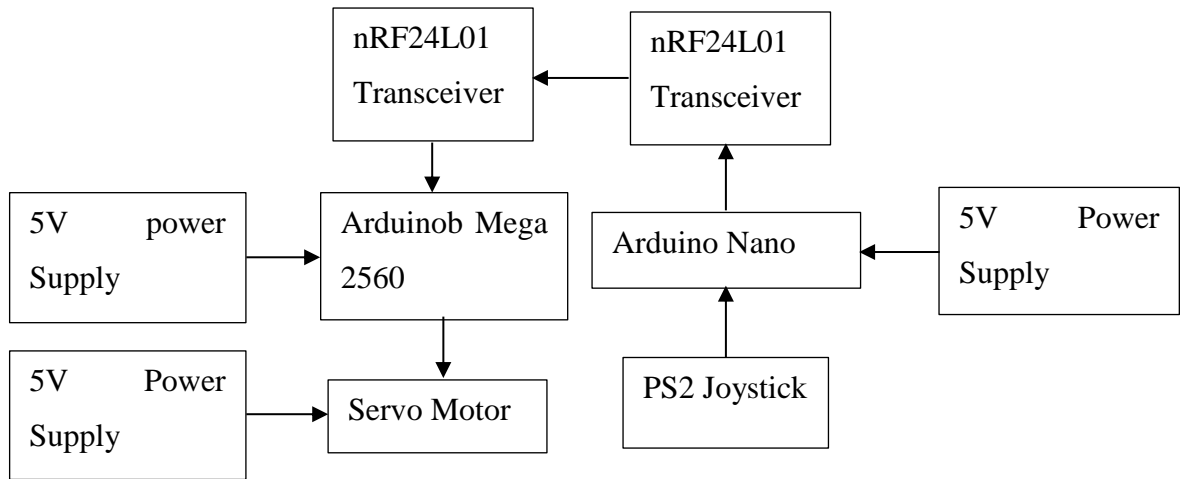


Figure 3.12 Relationship Between All Circuit for Quadruped Robot and Remote Controller.

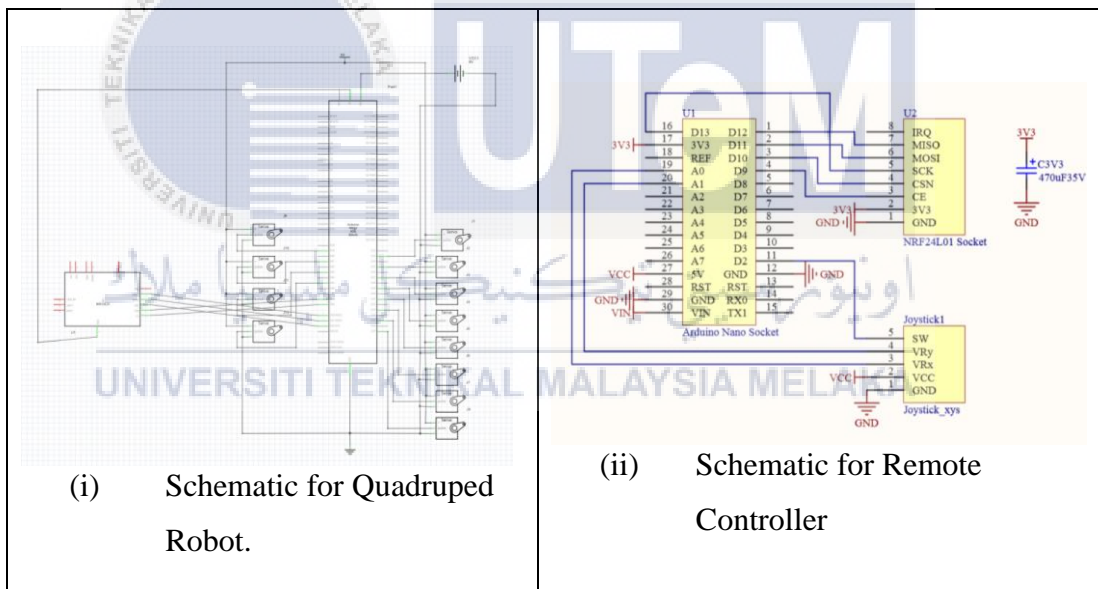


Figure 3.13 Schematic Drawing for The Circuit of Quadruped Robot and Remote Controller by Fritzing Software

### 3.6 Fabrication

This section will show the fabrication process of quadruped robot. After the CAD drawing for quadruped robot, the file will save into STL file. Then the STL file will open in a 3D printing software called CURA software to process the STL file into G-code file so that 3D printer is able to process the file. In CURA software, it is able to choose the infill percentage for the part and some other parameter for the 3D printer. After that, the G-code file need to transfer to 3D printer and insert the PLA filament to 3D printer and start printing the parts.

Then, the circuit connection for quadruped robot will solder on a perf board. The circuit should be simplified so that the circuit is easy connect and easy to debug when problems happen. The wires for live and neutral for all servo motors are connected together and power up by a 5V power supply. The signal wires from servo motors will connect to I/O port of Arduino Mega independently. For nRf24L01 transceiver, it will power up by Arduino Mega 5V output and CSN, CE, MOSI SCK and MISO port for the transceiver will connect to port 48 to port 52.

After the parts of quadruped robot are printed, the part will be assembled with servo motor and Arduino Mega. Figure 3.14 shows connection of perf board and Figure 3.15 shows the fabrication process of quadruped robot.

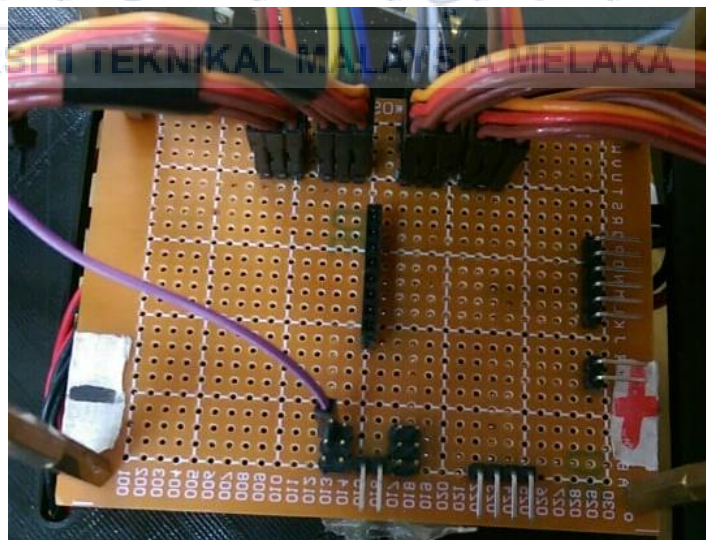


Figure 3.14 Connection of Perf Board.



Figure 3.15 Fabrication Process of quadraped robot.

### 3.7 Design of Walking Pattern for Quadraped Robot

In this section, the calculation of the degree for each servo will be shown and explained. After the calculation for the angle, the design of walking patterns will be shown together with explanation.

#### 3.7.1 Angle Calculation for Each Servo Motor

Before the design of walking pattern of quadraped robot, the calculation for the angle of servo motor is very important, so that it can be easy to code for the programming part.

The method used to calculate the angle of servo is conversion of cartesian coordinate to polar coordinate. There are some parameter needed for the calculation and main parameter are  $\alpha$ ,  $\beta$ , and  $\gamma$  which are the angle of femur, tibia and joint respectively. Figure 3.16 below shows the parameters needed to calculate angle of servo motor.

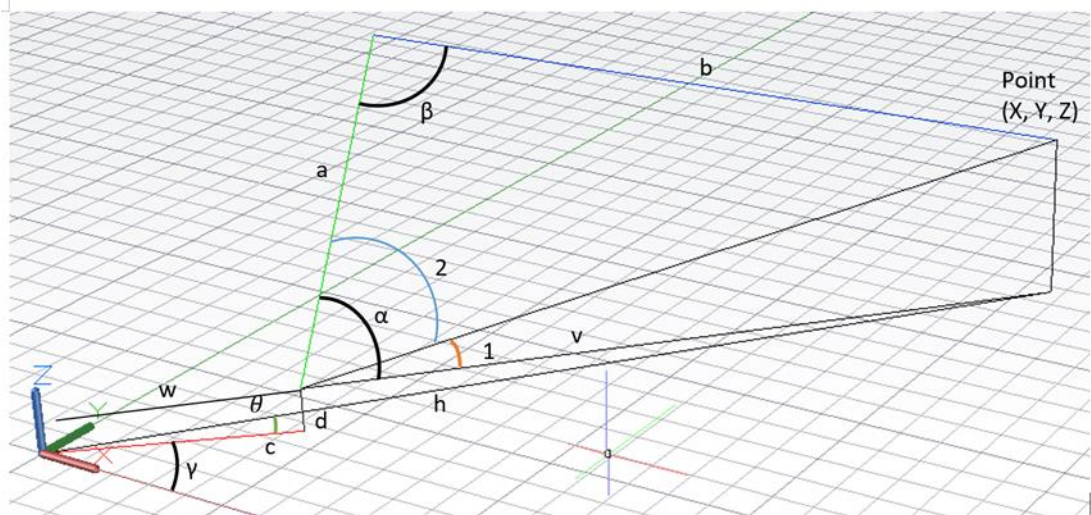


Figure 3.16 Parameters Needed to Calculate Angle of Servo Motor.

In Figure 3.16, it shows that the coordinate of point at (X, Y, Z). These are the known variables use to calculate the angle of  $\alpha$ ,  $\beta$ , and  $\gamma$ . Parameter a is the length of femur and parameter b is the length of tibia. For parameter c and d, they are the offset values at joint. Parameter h is the length from origin to the point in XY plane. The fomula to calculate value h is

$$h = \sqrt{(x^2 + y^2)} \quad (3-1)$$

Then calculate value of  $\theta$  by using the formula of,

$$\theta = \sin^{-1} \frac{d}{h} \quad (3-2)$$

Then, find the value of w and v by,

$$w = h \times \cos \theta \quad (3-3)$$

$$v = w - c \quad (3-4)$$

Then using the cosine rule formula, where

$$c^2 = a^2 + b^2 - 2ab \cos c \quad (3-5)$$

The value of  $\alpha$  and  $\beta$  can be calculated,

$$\angle 2 = \cos^{-1} \frac{a^2 + (z^2 + v^2) - b^2}{2 \times a \times \sqrt{z^2 + v^2}} \quad (3-6)$$

$$\angle \alpha = \angle 1 + \angle 2 \quad (3-7)$$



$$\angle\alpha = \tan^{-1} \frac{z}{v} + \cos^{-1} \frac{a^2 + (z^2 + v^2) - b^2}{2 \times a \times \sqrt{z + v^2}} \quad (3-8)$$

$$\angle\beta = \frac{a^2 + b^2 + (z^2 + v^2)}{2 \times a \times b} \quad (3-9)$$

Lastly, the value of  $\gamma$  can be calculated by,

$$\angle\gamma = \tan^{-1} \frac{y}{x} - \theta \quad (3-10)$$

After all these calculations, all the angles are found and can be used for the servo motors. These are the methods to transform a cartesian value to polar value. However, each leg has its own coordinate system, which is calculated independently. Therefore, every angle that used for the servo motor must be very careful.

### 3.7.2 Design for Sprawling Type and Lateral Walking Pattern

The design of sprawling type walking pattern and lateral walking pattern will start by the standing coordinate for each leg. After that, the coordinate of the step of each leg need to be decide and then to body movement also need to be consider. Table-3.9 show the table that coordinate needed to decide and record for quadruped robot's leg for lateral and sprawling walking pattern.

Table 3.9 The position of leg that needed to be decide for both sprawling and lateral walking pattern.

Sprawling type	Coordinate	Default		(x, y, z)	
		Position 1		(x, y, z)	
		Position 2		(x, y, z)	
	Step	Leg 1	Leg 2	Leg 3	Leg 4
	0				
	1				
	2				
	3				
	4				
	5				
6					
Lateral type	Coordinate	Default		(x, y, z)	
		Position 1		(x, y, z)	
		Position 2		(x, y, z)	

	Step	Leg 1	Leg 2	Step	Leg 1
	0				
	1				
	2				
	3				
	4				
	5				
	6				

### 3.8 Programing for Quaduped Robot and Wireless Remote Controller

This part will explain about the programming flow chart for the remote controller and quadruped robot. The microcontrollers used for this project are Arduino Mega and Arduino Nano. The Arduino language is basically a C/C++ language. After the coding is done in Arduino Software (IDE), it will compile into C/C++ and upload to microcontroller.



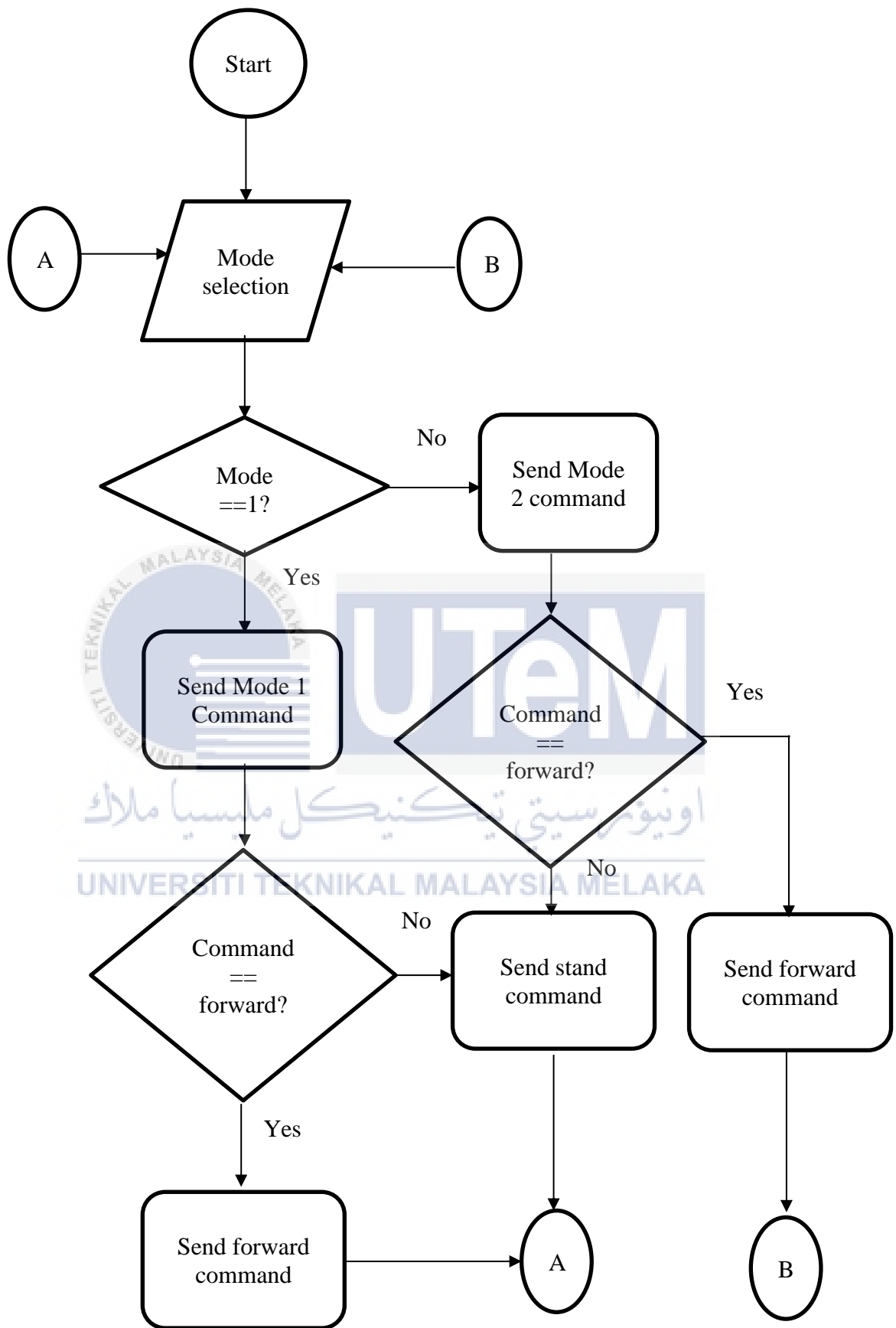


Figure 3.17 Flow chart for wireless remote controller.

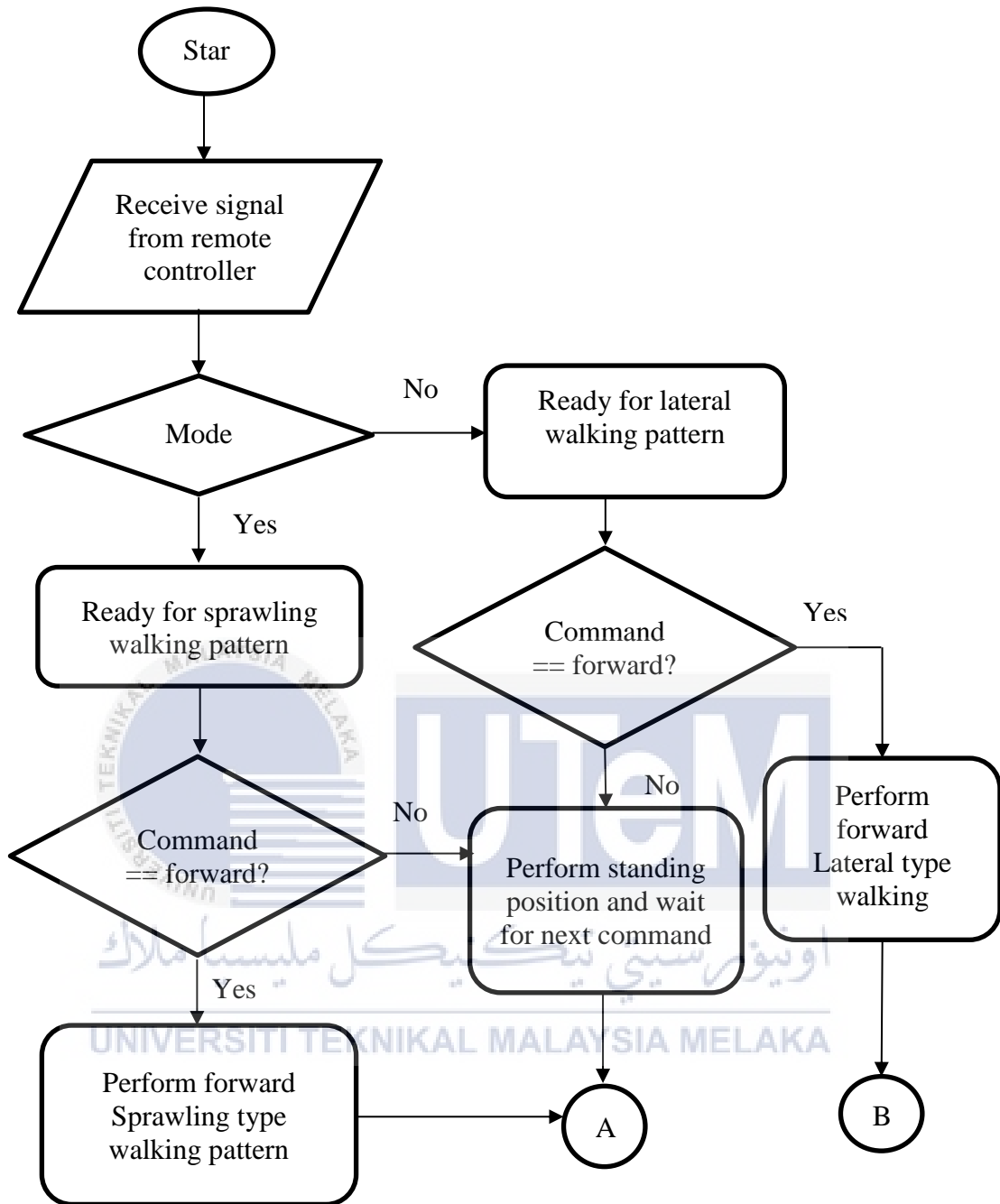


Figure 3.18 Flow chart for quadruped robot.

Figure 3.17 and Figure 3.18 shows the flow chart for wireless remote controller and quadruped robot respectively. Both of the flow chart is similar to each other. For the wireless controller, it will receive the command triggered by the user and send the command to quadruped robot to perform the movement that user wanted to perform. There are 2 choice of mode are available for the quadruped robot. The first mode is sprawling type walking pattern and the second type is lateral type of walking pattern. After the user change the mode, it will wait for user trigger the forward signal to call

the quadruped robot to perform forward movement, or else it will perform standing position to wait for the user give the next command.

### 3.9 Experiments

#### 3.9.1 Experiment 1: Test the error exists for servo motor

Objective: To test the error occur in actual position for each leg.

Material and Apparatus

1. Servo motors
2. Quadruped robot's leg printed by 3D printer
3. Protractor
4. Arduino board

Procedures:

1. Assembling the legs printed by 3d printer with the servo motors.
2. Connect servos motor to Arduino board.
3. Set the desire angle for the servo motor in the Arduino program and upload into Arduino board.
4. Power up Arduino board and servo motor by 18650 lithium ion battery.
5. Use protractor to measure the rotated angle of the servo motor.
6. Record the angle and calculate the error between desire angle and real angle presented by the servo motor.

Table 3.10 Data Require in Experiment 1

Desired angle	0	10	20	30	40	50	60	70	80	90
Exact angle										
Error										

Desired angle	100	110	120	130	140	150	160	170	180
Exact angle									
Error									

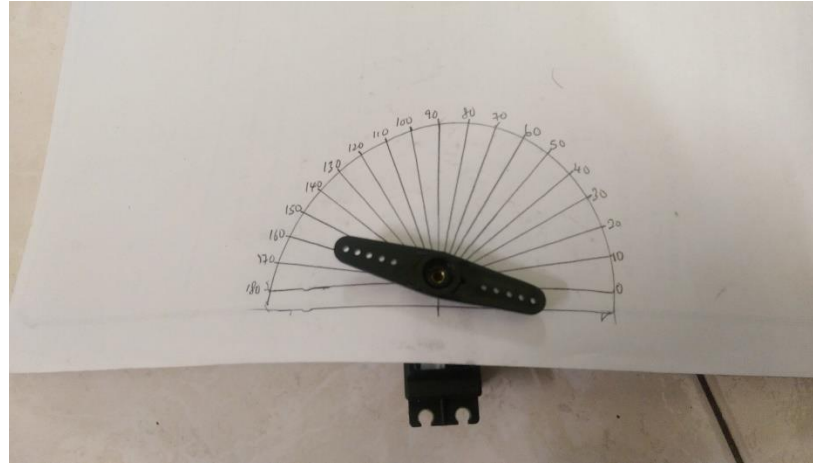


Figure 3.19 Experiment 1 Set Up

### 3.9.2 Experiment 2: Test the errors occurs for the movement of Quadruped Robot.

Objective: To test the movement of the robot see whether it can move as what we control.

Material and Apparatus

1. Assembled quadruped robot
2. Ruler and protractor
3. Masking tape

Procedures:

1. Select lateral-type walking mode for the quadruped robot.
2. Use masking tape to make a 2 m line on the flat surface ground.
3. Control the quadruped robot move forward along the line.

4. Measure the distance displace from the robot to the line.
5. Measure the angle between the line and the robot.
6. Calculate the error for the robot.
7. Repeat 1 to 6 for sprawling-type walking pattern for the quadruped robot.
8. Repeat 1 to 7 by changing the flat surface to grass surface.
9. Repeat 1 to 7 by changing the flat surface to tarred surface.

Table 3.11 Data Require in Experiment 2

Walking type	Flat surface		Grass surface		Tarred surface	
	Distance displace ( $x$ )	Angle ( $\theta$ )	Distance displace ( $x$ )	Angle ( $\theta$ )	Distance displace ( $x$ )	Angle ( $\theta$ )
lateral type						
Sprawling-type						

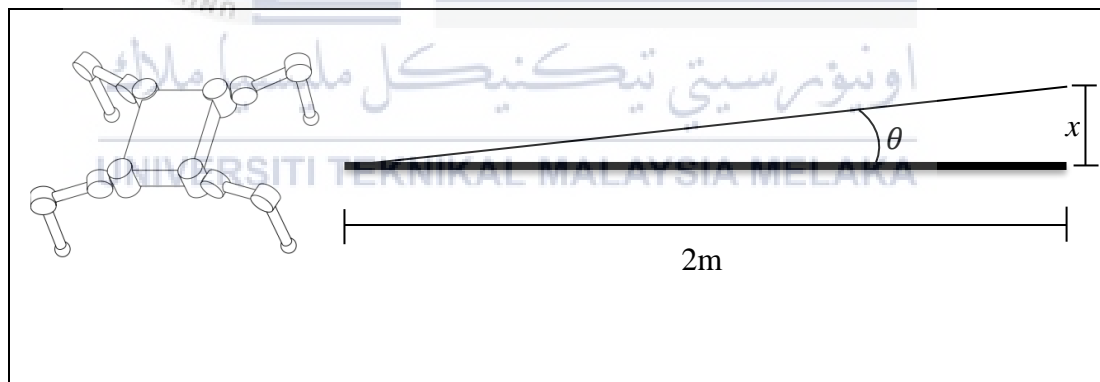


Figure 3.20 Experiment 2 Set Up

### 3.9.3 Experiment 3: Time taken for the robot passthrough different surface with different walking pattern

Objective: To calculate the speed of the robot when walking on the different tile surface in different walking pattern.

## Material and Apparatus

1. Assembled quadruped robot
2. Ruler
3. Masking tape
4. Stopwatch

## Procedures:

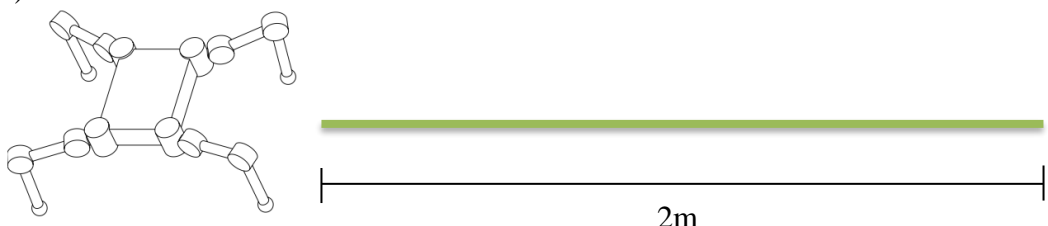
1. Select lateral type walking mode for the quadruped robot.
2. Use masking tape to make a marker at 2 m from the starting point on the flat surface ground.
3. Control the quadruped robot move forward and start counting the time when it started to move.
4. Stop the time when the robot reached the finish point.
5. Calculate the speed for the robot.
6. Repeat 1 to 5 in lateral-type walking for grass surface.
7. Repeat 1 to 5 in lateral-type walking for tarred surface.
8. Repeat 1 to 7 for sprawling-type walking pattern for the quadruped robot.

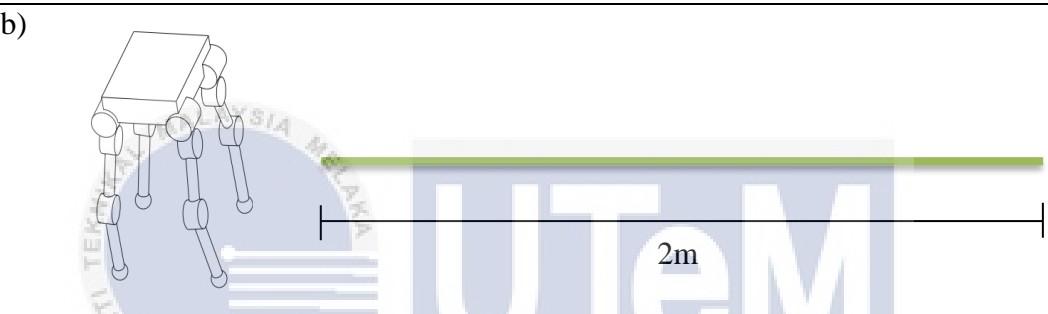
Table 3.12 Data Require in Experiment 3

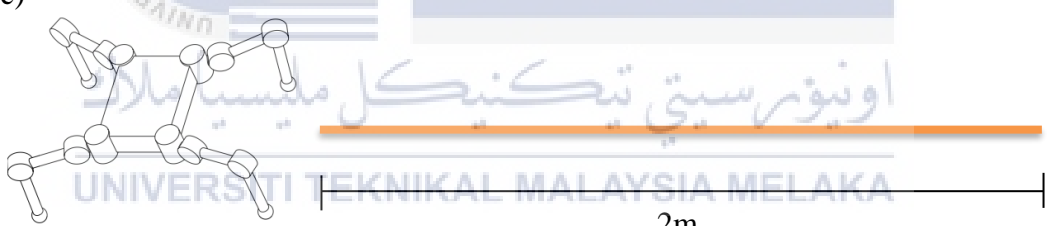
Walking type	Surface	Time (s)
Lateral type	Flat	
	Grass	
	Tarred surface	

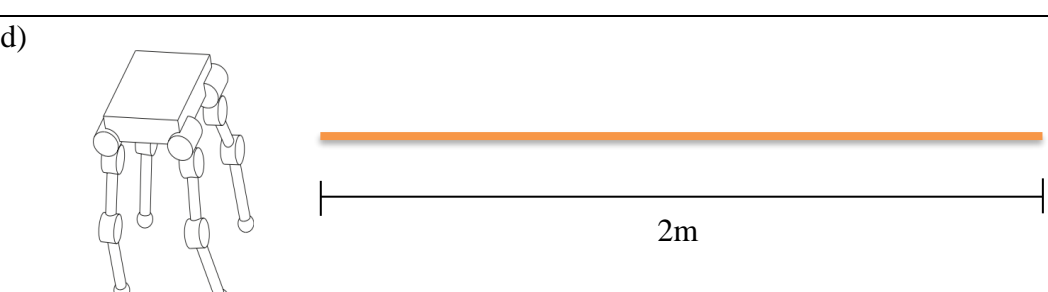


Sprawling-type	Flat	
	Grass	
	Tarred surface	

a)  Diagram of a sprawling-type robot with a green horizontal bar and a 2m scale bar.

b)  Diagram of a sprawling-type robot with a green horizontal bar and a 2m scale bar.

c)  Diagram of a sprawling-type robot with an orange horizontal bar and a 2m scale bar.

d)  Diagram of a sprawling-type robot with an orange horizontal bar and a 2m scale bar.

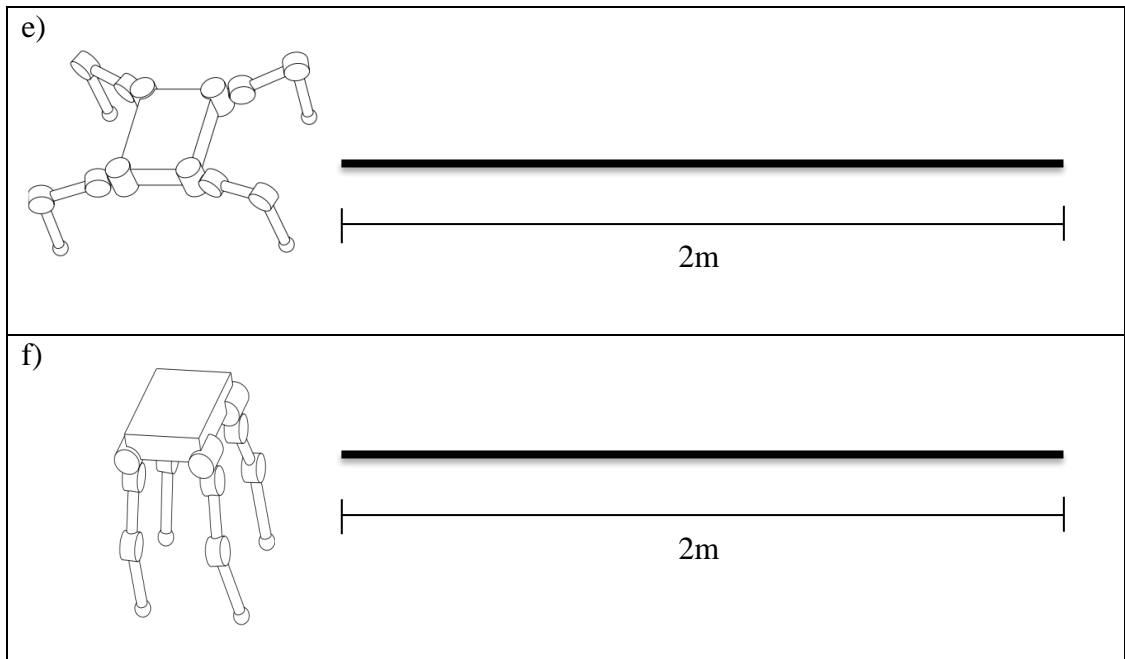


Figure 3.21 Experiment 3 Set Up



## CHAPTER 4

### RESULTS AND DISCUSSIONS

#### 4.1 Introduction

In this chapter, the result of this project will be shown, and the result from the experiments is analyzed and discussed. Basically, the results shown will be the final design for the quadruped robot, the fabricated quadruped robot, the calibration of quadruped robot before experiment, the result of walking pattern design and the experiment results mention in methodology.

#### 4.2 Result for Design and Construction of Quadruped Robot

This is the section to shows the conceptual design and the finalize design of the quadruped robot. The comparison between conceptual design and finalize design will be discussed and the calibration for the quadruped robot will be shown before carrying out the experiments in this project.

##### 4.2.1 Conceptual Design

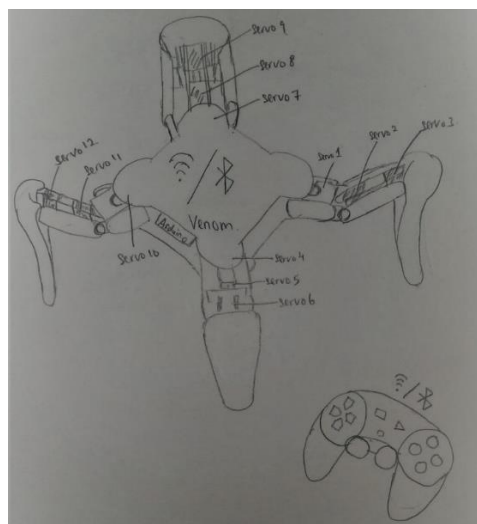


Figure 4.1 Design of quadruped robot by using hand drawing.

Figure 4.1 shows the draft design of quadruped robot which drawn by hand draft. The draft shows the robot consist of 12 servo motor, and it is controlled by a joystick controller. The communication between the joystick controller and the quadruped robot is either Bluetooth or Wi-Fi. Since this robot need to perform mammal-type and sprawling-type walking pattern, hence the main part of the robot will be the servo motor 1,4,7, and 10. If the robot wants to perform mammal-type walking pattern, servo motor 1 and 7 need to turn for  $180^\circ$ , while servo motor 4, and 10 need to turn for  $0^\circ$ . When the robot wants to perform a sprawling-type waking pattern, the servo motor 1 and 7 need to turn for  $135^\circ$ , while servo motor 4 and 10 need to turn for  $45^\circ$ . The microprocessor of the robot is Arduino, which is commonly used and easy to program since it is open source. It will connect to Bluetooth or Wi-Fi module to communicate with the joystick controller. On the other hand, joystick controller can change the mode for walking pattern by clicking a button. It can also add other mode for the robot since the joystick controller has many button to set different command.

#### 4.2.2 Finalize Design



Figure 4.2 Final design of quadruped robot in SolidWorks.

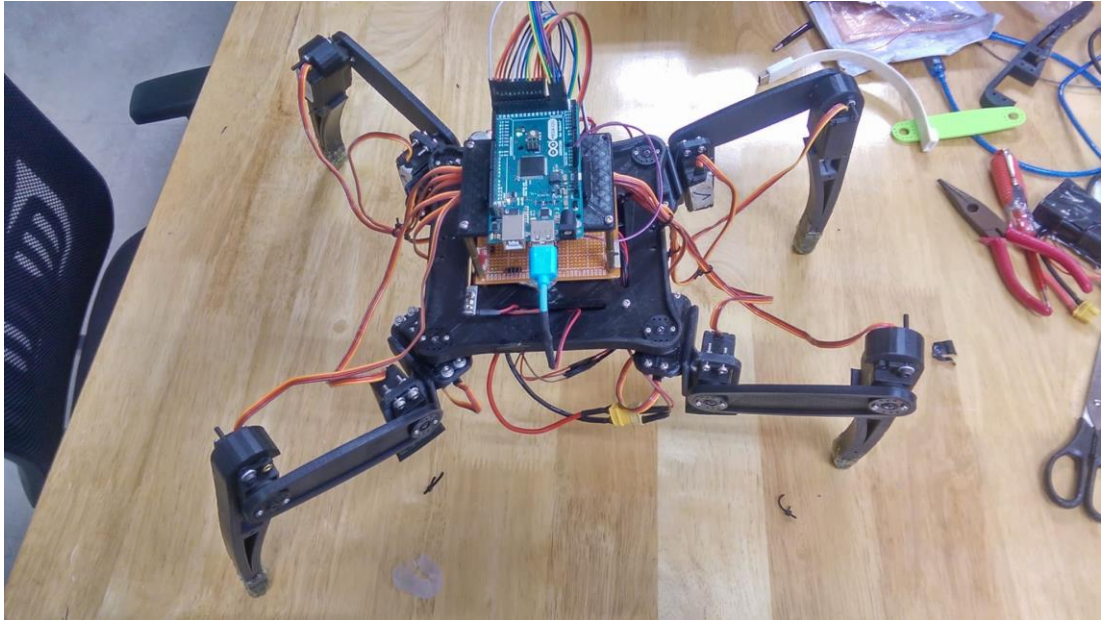


Figure 4.3 Fabricated quadruped robot.

Figure 4.2 and Figure 4.3 show the finalized design for the quadruped robot. The fabricated hardware result is very close to the finalized design of the quadruped robot. The 3D printed body of the quadruped robot is able to withstand the weight of itself. This is the result that achieves the expectation and can perform lateral and sprawling type walking patterns.

However, there is a hardware problem that affects the performance of the quadruped robot, which is the servo motors and servo motor horns. The contact between the servo motors and servo motor horns is not tight enough and causes a shaky problem when an external force is applied to the quadruped robot. Besides that, one of the servo motors is not stable in its performance. Sometimes this servo motor malfunctions and sometimes it can work very well. To prevent these problems from happening, replace the old servo motor horn and replace the servo motor that has a problem.

### 4.2.3 Calibration for Quadruped Robot

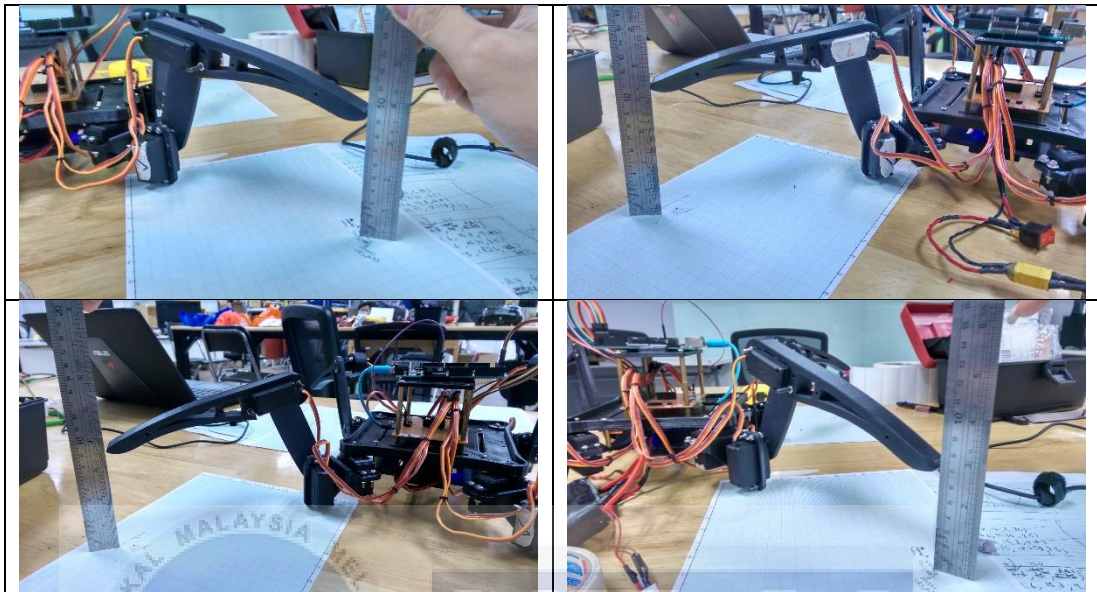


Figure 4.4 Calibration of quadruped robot's legs.

Figure 4.4 shown the method to calibrate the legs of quadruped robot. After the quadruped robot is assembled, the calibration process is a must before carrying out any experiments. This is because error may happen to mechanical connection during the installation. Therefore, quadruped robot needs to go through calibration process to make sure the accuracy of the quadruped robot.

The way to calibrate is give a reference coordinate for the legs and run the “Adjust” code written in Arduino. Then, measure the real coordinate for each leg and key in the value that measured in to the coding. After that run the “verify” code written in Arduino, then the microprocessor will calculate the error and calibrate it to the desire coordinate. Lastly, measure the coordinate for the calibrated quadruped robot's legs. If the result is close to the desire coordinate, that means the calibration is successful, or else carry out the calibration process until successful.

#### 4.2.4 Walking Pattern Result

Table 4.1 Walking pattern for 1 cycle for sprawling and lateral walking pattern

Sprawling type	Coordinate	Default		(110, 0, -140)	
		Position 1		(110, 80, -140)	
		Position 2		(110, 160, -140)	
	Step	Leg 1	Leg 2	Leg 3	Leg 4
	0	Position 1	Position 1	Default	Default
	1	Position 1	Position 1	Default	Position 2
	2	Default	Position 2	Position 1	Position 1
	3	Default	Default	Position 1	Position 1
	4	Position 2	Default	Position 1	Position 1
	5	Position 1	Position 1	Position 2	Default
6	Position 1	Position 1	Default	Default	
Lateral type	Coordinate	Default		(0, 40, -160)	
		Position 1		(0, 60, -160)	
		Position 2		(0,120, -160)	
	Step	Leg 1	Leg 2	Step	Leg 1
	0	Position 1	Position 1	Default	Default
	1	Position 1	Position 1	Default	Position 2
	2	Default	Position 2	Position 1	Position 1
	3	Default	Default	Position 1	Position 1
	4	Position 2	Default	Position 1	Position 1
	5	Position 1	Position 1	Position 2	Default
6	Position 1	Position 1	Default	Default	

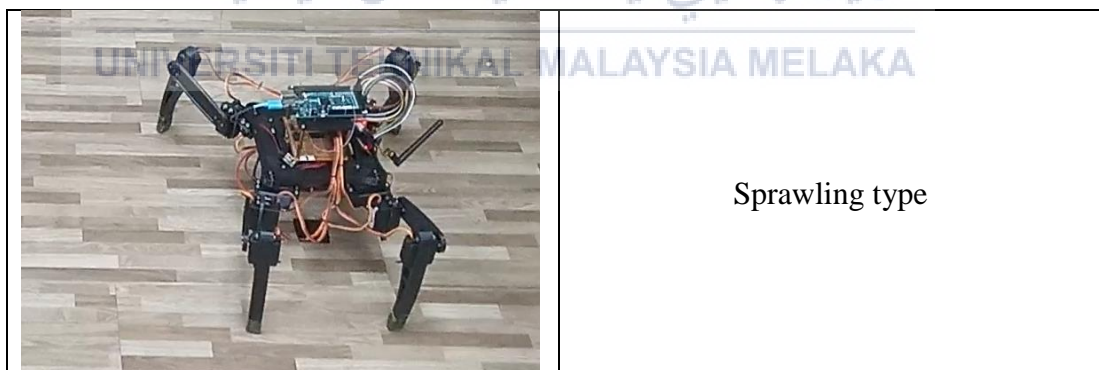




Figure 4.5 Sprawling type and lateral type of walking pattern.

Table 4.1 shows the walking pattern for 1 cycle for both sprawling and lateral walking pattern. The cycle for each leg is similar for both walking pattern. The main different between these two types of walking pattern is the position of the leg. For sprawling type waling pattern, its standing surface area is bigger than lateral type walking pattern. However, the height of the quadruped robot in lateral type walking pattern is higher than sprawling type. By comparing the differences, it can be concluded that sprawling type walking pattern is a more stable walking pattern compare to lateral type of walking pattern. This is because sprawling type walking pattern has a larger base and lower center of gravity (CG).

### 4.3 Experiment Results

#### 4.3.1 Accuracy test for servo motors for each leg of the quadruped robot

Table 4.2 Exact angle and error occur on servo motor

Desired angle	0	10	20	30	40	50	60	70	80	90
Exact angle	0	10	20	30	40	50	59	68	78	85
Error	0%	0%	0%	0%	0%	0%	1.67%	2.86%	2.5%	5.56%
Desired angle	100	110	120	130	140	150	160	170	180	
Exact angle	95	105	115	125	133	140	150	160	170	
Error	5%	4.55%	4.17%	3.85%	5%	6.67%	6.25%	5.88%	5.56%	



Table 4.2 shown the error that occur on servo motors. The error is calculated by the different between desired angle and the exact angle measured divide by desired angle multiply by 100%.

$$\text{Error} = \frac{\text{desired angle} - \text{excat angle}}{\text{desired angle}} \times 100\% \quad (4-1)$$

The received result is used to plot the graph in Figure4.6 below.

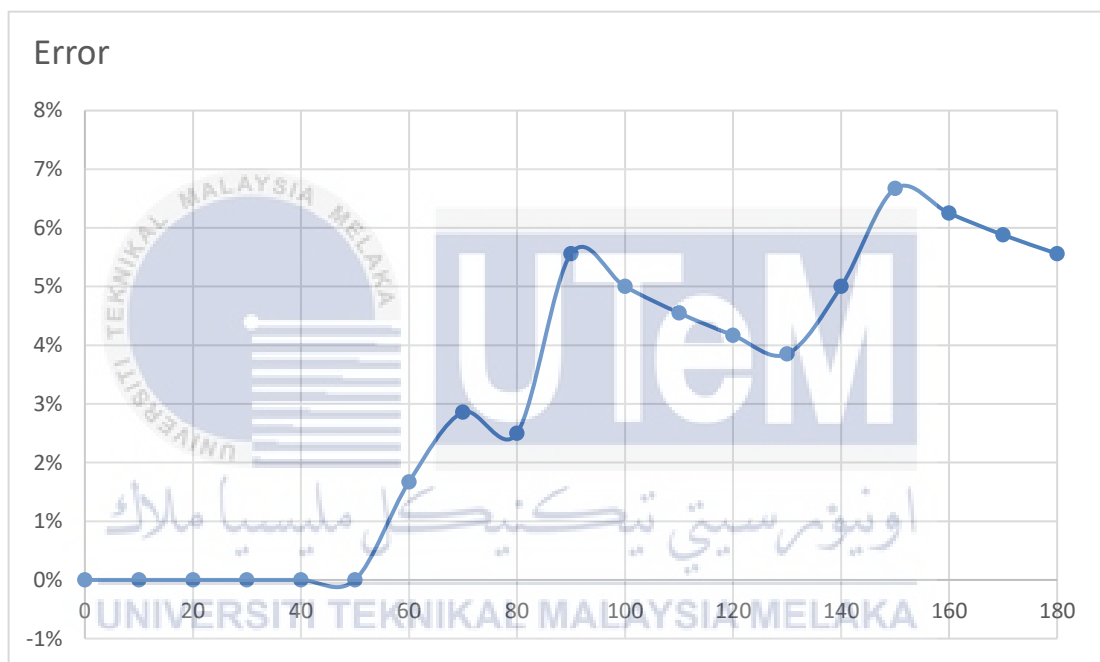


Figure 4.6 Graph of error occur (%) against the desire angle.

Base on the calculated error, it shows that the error of servo motors is below 10%, which is similar to the result in the data sheet given. The accuracy of servo motors below 40° is 100%. The errors occur when the servo motor rotate more than 40° and the peak error occur when the servo motor rotate to 150° which is 6.67% of error. Through this experiment, the expected result for the error occur for walking pattern will be around 10 % since the accuracy of the quadruped robot is depend on the accuracy of servo motors.

### 4.3.2 Accuracy test for quadruped robot while walking in different walking pattern and different movements.

This experiment will calculate the error occur on the robot while it is walking in different walking pattern and different movement. Figure 4.6 below shows the experiment carryout on different surfaces.



Figure 4.7 The experiment carried out on different surface.

Table 4.3 The offset distances and angles when quadruped walk through 2 m of different surface environments.

Walking type	Flat surface		Grass surface		Tarred surface	
	Offset displace (x/ cm)	Angle ( $\theta$ )	Offset displace (x/ cm)	Angle ( $\theta$ )	Offset displace (x/ cm)	Angle ( $\theta$ )
lateral type	26	7.4°	Fail	-	39	11.03°
Sprawling-type	15	4.28°	63	17.48°	28	7.97°

Table 4.3 show the offset distances and angles when quadruped walk through 2 m of different surface environments. The angle is calculated by using the formula

$$\text{angle}(\theta) = \tan^{-1} \frac{\text{offset distance}(x)}{200\text{cm}} \quad (4-2)$$

The offset distance (x) is then used to calculate the error by the formula below,

$$\text{Error} = \frac{\text{offset distance}(x)}{200\text{cm}} \times 100\% \quad (4-3)$$

Table 4.4 The offset distances and calculated error for quadruped walk through 2 m of different surface environments.

Walking type	Flat surface		Grass surface		Tarred surface	
	Offset displace (x/ cm)	Error (%)	Offset displace (x/ cm)	Error (%)	Offset displace (x/ cm)	Error (%)
lateral type	26	13	Fail	-	39	19.5
Sprawling-type	15	7.5	63	31.5	28	14

Base on the result on Table 4.4, it shows that the error occurs for sprawling type walking pattern is relatively smaller than a lateral type walking pattern. The error

occurs in flat surface for sprawling type of walking pattern is 7.5%, and the error occurs in flat surface for lateral walking pattern is 13%. However, the error increased for both type of walking pattern on grass surface. The error occur for sprawling type walking pattern is 31.5% and the lateral type of walking pattern cannot complete the task on grass surface. For the tarred surface the error for sprawling type walking pattern is 14% while lateral type walking pattern is 19.5%.

Based to the result, both type of walking pattern can perform well in a flat surface and error occur is below 15%, The error start to increase for both walking pattern when walking a tarred surface. This is because the friction on a tarred surface is higher than flat surface. The worst result for both walking pattern is walking on grass surface. This is because grass surface is an uneven surface and the quadruped robot is trapped by some grass when walking on grass. Lateral walking pattern cannot complete the task on grass surface because the centre of gravity (CG) is higher in this walking pattern and the surface area of its base is smaller. It was fall down when perform a few step movements on the grass surface.

### 4.3.3 Speed test for the quadruped robot in different type of walking pattern on different surfaces.

Table 4.5 Time taken for sprawling type and lateral type of walking pattern walk on different surfaces for 2 meters.

Walking type	Surface	Time (s)
Sprawling-type	Flat	37.7
	Grass	59.8
	Tarred surface	48.5
Lateral type	Flat	39.2
	Grass	-
	Tarred surface	50.4

Table 4.5 shown the time taken for sprawling type and lateral type of walking pattern walk on different surfaces for 2 meters. The result is then use for calculating the speed for both type of walking pattern on different surfaces. The formula to calculate the speed is,

$$Speed = \frac{2m}{time} \quad (4-4)$$

Table 4.6 The speed calculated for sprawling type and lateral type of walking pattern walk on different surfaces for 2 meters.

Walking type	Surface	Time (s)	Speed (m/s)
Sprawling-type	Flat	28.4	0.070
	Grass	48.5	0.041
	Rock surface	37.7	0.053
Lateral type	Flat	39.2	0.051
	Grass	-	-
	Rock surface	50.4	0.040

The calculated speed for both sprawling type and lateral type of walking pattern on different surfaces is shown on the Table 4.6. In general, the speed for sprawling type walking pattern is higher than lateral type walking pattern. This is because the step for lateral type walking pattern is smaller than sprawling type walking pattern. The performance of quadruped robot in a flat surface is the best for both sprawling type and lateral type of walking pattern. The speed on a flat surface for sprawling and lateral type of walking pattern are 0.070m/s and 0.051m/s respectively. For the rock surface, the speed of sprawling type walking pattern is 0.053m/s while lateral type walking pattern is 0.040m/s. The performance is on grass surface is the worst for both walking pattern. Sprawling type walking pattern only reach the speed on 0.041m/s when walking on grass surface, while lateral type walking pattern totally cannot complete the task.

## CHAPTER 5

### CONCLUSION AND RECOMMENDATIONS

#### 5.1 Conclusion

In conclusion, this design and construction of a 4-legged robot in various surface environment is success because all the objectives are achieved. The first objective is design and construct a quadruped robot, and it is successfully been develop. The quadruped is able to control by a wireless remote controller and is able to perform 2 walking patterns which are sprawling type and lateral type of walking pattern. This is another objective that achieved in this project. The last objective is to analyze the walking pattern of the quadruped robot. The quadruped robot is analyzed in the accuracy and the speed for both sprawling and lateral type of walking pattern.

#### 5.2 Future Works

There are some issues that needed to be concern for the project. The first thing is the design of the quadruped robot. The dimension of the servo motor needs to be measured properly so that the 3D printed part for the robot can fit the servo motors. Besides the circuit of the robot also need to be design properly, so that the servo motors have enough power to power up all 12 servo motors. Then the calibration of robot needs to be done before carrying out the experiments. It is a very important part for quadruped robot, because it will affect the accuracy of robot.

## REFERENCES

- [1] C. Hilary Whiteman, "Japan officials: Missing bolts may have caused tunnel collapse," *CNN*. [Online]. Available: <https://edition.cnn.com/2012/12/03/world/asia/japan-tunnel-collapse-bolts/index.html>.
- [2] R. P. Marc Raibert, Kevin Blankespoor, Gabriel Nelson, "BigDog, the Rough-Terrain Quadruped Robot Marc," *Proc. 17th IFAC World Congr.*, pp. 1–8, 2008.
- [3] S. Kitano, S. Hirose, A. Horigome, and G. Endo, "TITAN-XIII: sprawling-type quadruped robot with ability of fast and energy-efficient walking," *ROBOMECH J.*, vol. 3, no. 1, pp. 1–16, 2016.
- [4] L. Wang, W. Du, X. Mu, X. Wang, G. Xie, and C. Wang, "A geometric approach to solving the stable workspace of quadruped bionic robot with hand-foot-integrated function," *Robot. Comput. Integr. Manuf.*, vol. 37, pp. 68–78, 2016.
- [5] B. Na and K. Kong, "Design of a One Degree-of-Freedom Quadruped Robot Based on a Mechanical Link System: Cheetaroid-II," *IFAC-PapersOnLine*, vol. 49, no. 21, pp. 409–415, 2016.
- [6] I. Gonzalez-Luchena, A. G. Gonzalez-Rodriguez, A. Gonzalez-Rodriguez, C. Adame-Sanchez, and F. J. Castillo-Garcia, "A new algorithm to maintain lateral stabilization during the running gait of a quadruped robot," *Rob. Auton. Syst.*, vol. 83, pp. 57–72, 2016.
- [7] M. Li, Z. Jiang, P. Wang, L. Sun, and S. S. Ge, "Control of a quadruped robot with bionic springy legs in trotting gait," *J. Bionic Eng.*, vol. 11, no. 2, pp. 188–198, 2014.
- [8] M. C. García-López, E. Gorrostieta-Hurtado, E. Vargas-Soto, J. M. Ramos-Arreguín, A. Sotomayor-Olmedo, and J. C. M. Morales, "Kinematic analysis for trajectory generation in one leg of a hexapod robot," *Procedia Technol.*, vol. 3, pp. 342–350, 2012.

- [9] V. G. Loc, I. M. Koo, D. T. Tran, S. Park, H. Moon, and H. R. Choi, "Improving traversability of quadruped walking robots using body movement in 3D rough terrains," *Rob. Auton. Syst.*, vol. 59, no. 12, pp. 1036–1048, 2011.
- [10] H. C. Shamsudin, M. A. Ayob, W. N. Wan Zakaria, and M. F. Zakaria, "Modeling and Simulation for One Leg of Quadruped Robot Trajectory," *Appl. Mech. Mater.*, vol. 826, no. February, pp. 140–145, 2016.
- [11] H. Nakashima, M. Tokuda, H. Yamamoto, and M. Funakoashi, "Load Distribution and Body Angle Measurement in Static Walking of a Quadruped Walking Robot," *IFAC Proc. Vol.*, vol. 33, no. 29, pp. 95–99, 2000.
- [12] T. Zielinska and J. Heng, "Mechanical design of multifunctional quadruped," *Mech. Mach. Theory*, vol. 38, no. 5, pp. 463–478, 2003.
- [13] L. C. Hale, *Principles and Techniques for Designing Precision Machines*, no. February. 1999.
- [14] C. D. Remy *et al.*, "Walking and crawling with ALoF: a robot for autonomous locomotion on four legs," *Ind. Robot An Int. J.*, vol. 38, no. 3, pp. 264–268, 2011.
- [15] G. S. Sukhatme, "The Design and Control of a Prototype Quadruped Microrover," *Robot. Res. Lab. Dep. of Computer Sci. Inst. Robot. Intell. Syst.*, vol. 220, p. 25, 1997.
- [16] C. Semini, N. G. Tsagarakis, B. Vanderborcht, Y. Yang, and D. G. Caldwell, "HyQ – Hydraulically Actuated Quadruped Robot Hopping Leg Prototype.pdf," *2nd Bienn. IEEE/RAS-EMBS Int. Conf. Biomed. Robot. Biomechatronics*, pp. 593–599, 2008.
- [17] M. H. Raibert, "Legged robots," *Commun. ACM*, vol. 29, no. 6, pp. 499–514, 1986.
- [18] C. P. Santos and V. Matos, "Gait transition and modulation in a quadruped robot: A brainstem-like modulation approach," *Rob. Auton. Syst.*, vol. 59, no. 9, pp. 620–634, 2011.
- [19] R. B. Mcghee and A. A. Fkask, "On the Stability Properties of Quadruped



- Creeping Gaits,” *Math. Biosci.* 3, vol. 351, pp. 331–351, 1968.
- [20] K. Inagaki and H. Kobayashi, “A Gait Transition for Quadruped Walking Machine,” *Proceeding of the 1993 IEEE/RSJ Int. Conf. Intell. Robot. Syst.*, vol. 00, no. c, pp. 525–531, 1993.
- [21] F. Hardarson, “Stability analysis and synthesis of statically balanced walking for quadruped robots,” Royal Institute of Technology, KTH, 2002.
- [22] M. Buehler, R. Playter, and M. Raibert, “Robots Step Outside,” *Int. Symp. Adapt. Motion Anim. Mach.*, no. September 2005, 2005.
- [23] R. Playter, M. Buehler, M. Raibert, and B. Dynamics, “BigDog,” *Proc. SPIE 6230*, pp. 896-901, no. April 2006, 2006.

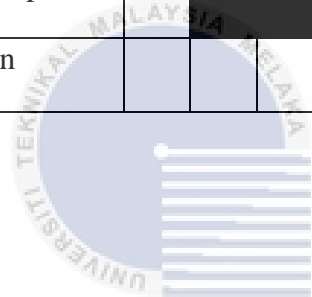


## APPENDICES

### APPENDIX A GANTT CHART

Activity	WEEK													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Briefing														
Registration														
Gantt Chart														
Literature Review														
Introduction														
Objective														
Scope														
Problem Statement														
Methodology														
Preliminary Result														
Ready for Slide														
Finish Slide														
Presentation														
Plan FYP 2														

Activity	WEEK													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Technical Drawing														
Circuit Diagram														
Fabrication														
Assemble														
Programming														
Testing														
Analysis														
Technical Report														
Presentation														



اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

## APPENDIX B CODING FOR QUADRUPED ROBOT.

```
#include <FlexiTimer2.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <printf.h>
#include <RF24.h>
#include <RF24_config.h>
#include <EEPROM.h>
#include <Servo.h>
/* Have a serial monitor to check error*/
#define SERIAL_PRINT
#define SERIAL_PRINT_WIFI
#define SERIAL_PRINT_WALKING
/* Installation and Adjustment -----*/
#define INSTALL //uncomment only this to install the robot
#define ADJUST //uncomment only this to adjust the servos
#define VERIFY //uncomment only this to verify the adjustment
const float adjust_site[3] = { 160, 160, 120 };
const float real_site[4][3] = { { 183, 153, 65 }, { 164.5, 159, 109 }, { 165.5, 157.5,
95 }, { 165, 169, 89 } };
/*servos -----*/
Servo servo[4][3];
#define servos ports
const int servo_pin[4][3] = { {25, 23, 27}, {31, 29, 33}, {37, 39, 35}, {43, 45, 41} };
/* Wireless communication -----*/
#define RF24 for nRF24L01
RF24 radio(48, 49); //CNS,CE
#define RF24 transmit address
const byte address[6] = "00001";
```

```

/* Size of the robot -----*/
const float length_a = 100;
const float length_b = 160;
const float length_c = 35.5;
const float length_d = 10;
const float length_side = 66;
const float z_absolute = -45;
/* Constants for movement -----*/
const float z_default_1 = -140, z_default_2 = -140, z_default_3 = -160;
const float z_boot = z_absolute, z_up_1 = -10, z_up_2 = -90, z_up_3 = -90;
const float x_default_1 = 110, x_default_2 = 110, x_default_3 = 0, x_offset = 0;
const float y_start_1 = 0, y_start_2 = 0, y_start_3 = 40;
const float y_step_1 = 80 , y_step_2 = 80, y_step_3 = 60, y_offset = 30;
/* variables for movement -----*/
volatile float site_now[4][3]; //real-time coordinates of the end of each leg
volatile float site_expect[4][3]; //expected coordinates of the end of each leg
float temp_speed[4][3]; //each axis' speed, needs to be recalculated before
each movement
float move_speed; //movement speed
float speed_multiple = 1; //movement speed multiple
float x_default = x_default_1;
float y_start = y_start_1;
float y_step = y_step_1;
float z_up = z_up_1;
float z_default = z_default_1;
const float spot_turn_speed = 2;
const float leg_move_speed = 3;
const float leg_trot_speed = 4;
const float body_move_speed = 1.5;
const float stand_seat_speed = 1;
volatile int rest_counter; //+1/0.02s, for automatic rest
const int wait_rest_time = 3 * 50; //3s*50Hz, the time wait for automatic rest
//functions' parameter

```

```

const float KEEP = 255;
//define PI for calculation
const float pi = 3.1415927;
/* Constants for turn -----*/
//temp length
const float temp_a = sqrt(pow(2 * x_default + length_side, 2) + pow(y_step, 2));
const float temp_b = 2 * (y_start + y_offset + y_step) + length_side;
const float temp_c = sqrt(pow(2 * x_default + length_side, 2) + pow(2 * y_start +
y_offset + y_step + length_side, 2));
const float temp_alpha = acos((pow(temp_a, 2) + pow(temp_b, 2) - pow(temp_c, 2))
/ 2 / temp_a / temp_b);
//site for turn
const float turn_x1 = (temp_a - length_side) / 2;
const float turn_y1 = y_start + y_offset + y_step / 2;
const float turn_x0 = turn_x1 - temp_b * cos(temp_alpha);
const float turn_y0 = temp_b * sin(temp_alpha) - turn_y1 - length_side;
/* -----*/
/* Data receive from controller -----*/
int M = 1;
struct package {
int mode = 1;
int leg = 0;
// int servo_value[4][3] = {{90, 90, 90}, {90, 90, 90}, {90, 90, 90}, {90, 90, 90}};
int z_axis = 0;
int Mode_1_CMD = 0; // 0 = stand, 1 = forward, 2 = backward, 3 = turn_left, 4 =
turn_right;
int Move_speed = 0; // add move speed of the robot
int Mode_2_CMD = 0; // 0 = stand, 1 = forward, 2 = backward, 3 = turn_left, 4 =
turn_right;
int Mode_3_CMD = 0; // 0 = stand, 1 = forward, 2 = backward
};
typedef struct package Package;
Package data;

```

```

void setup() {
  Serial.begin(9600);
#ifdef INSTALL
  //initialize all servos
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      servo[i][j].attach(servo_pin[i][j]);
      delay(100);
    }
  }
  Serial.print("Installation Complete");
  while (1);
#endif
#ifdef ADJUST
  adjust();
  Serial.println("Adjust done");
  while (1);
#endif
#ifdef VERIFY
  verify();
  Serial.println("Verify done");
  while (1);
#endif
  Serial.println("Robot ready to start!!!");
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_HIGH);
  radio.startListening();
  Serial.println("Radio listening started");
  //initialize default parameter
  set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_boot);

```

```

set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_boot);
set_site(2, x_default + x_offset, y_start + y_offset, z_boot);
set_site(3, x_default + x_offset, y_start + y_offset, z_boot);
for (int i = 0; i < 4; i++)
{
  for (int j = 0; j < 3; j++)
  {
    site_now[i][j] = site_expect[i][j];
  }
}
//start servo service
FlexiTimer2::set(20, servo_wifi_service);
FlexiTimer2::start();
Serial.println("Servo service started");
//initialize servos
for (int i = 0; i < 4; i++)
{
  for (int j = 0; j < 3; j++)
  {
    servo[i][j].attach(servo_pin[i][j]);
    delay(100);
  }
}
Serial.println("Servos initialized");
Serial.println("Robot initialization Complete");
stand(data.z_axis);
Serial.println("Robot Standing");
}
void loop()
{
#ifdef INSTALL
  while (1);
#endif
}

```



```

#ifdef ADJUST
    while (1);
#endif
#ifdef VERIFY
    while (1);
#endif

if (data.mode == 1)
{
    x_default = x_default_1;
    y_start = y_start_1;
    y_step = y_step_1;
    z_up = z_up_1;
    z_default = z_default_1;
    Serial.println("Mode 1");
    if (M != 1)
    {
        sit();
        move_speed = stand_seat_speed;
        set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_boot);
        set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_boot);
        set_site(2, x_default + x_offset, y_start + y_offset + y_step, z_boot);
        set_site(3, x_default + x_offset, y_start + y_offset + y_step, z_boot);
        wait_all_reach();

        set_site(2, x_default + x_offset, y_start + y_offset, z_boot);
        set_site(3, x_default + x_offset, y_start + y_offset, z_boot);
        wait_all_reach();

        M = 1;
    }
    switch (data.Mode_1_CMD)
    {
        case 0:
            stand(data.z_axis);
            break;
    }
}

```

```

case 1:
    Serial.println("Step forward");
    step_forward(1, data.z_axis, data.Move_speed);
    break;
case 2:
    Serial.println("Step backward");
    step_back(1, data.z_axis, data.Move_speed);
    break;
case 3:
    Serial.println("Turn left");
    turn_left(1, data.z_axis, data.Move_speed);
    break;
case 4:
    Serial.println("Turn right");
    turn_right(1, data.z_axis, data.Move_speed);
    break;
}
}
else if (data.mode == 2)
{
    x_default = x_default_2;
    y_start = y_start_2;
    y_step = y_step_2;
    z_up = z_up_2;
    z_default = z_default_2;
    Serial.println("Mode 2");
    if (M != 2)
    {
        sit();
        move_speed = stand_seat_speed;
        set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_boot);
        set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_boot);
        set_site(2, x_default + x_offset, y_start + y_offset + y_step, z_boot);
    }
}

```

```

set_site(3, x_default + x_offset, y_start + y_offset+ y_step, z_boot);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_boot);
set_site(3, x_default + x_offset, y_start + y_offset, z_boot);
wait_all_reach();
M = 2;
}
switch (data.Mode_2_CMD)
{
case 0:
    Serial.println ("Stand");
    stand(data.z_axis);
    break;
case 1:
    Serial.println("Step forward");
    step_forward_M2(1, data.z_axis, data.Move_speed);
    break;
case 2:
    Serial.println("Step backward");
    step_backward_M2(1, data.z_axis, data.Move_speed);
    break;
case 3:
    Serial.println("Turn left");
    turn_left_M2(1, data.z_axis, data.Move_speed);
    break;
case 4:
    Serial.println("Turn right");
    turn_right_M2(1, data.z_axis, data.Move_speed);
    break;
}
}
if (data.mode == 3)
{

```

```

x_default = x_default_3;
y_start = y_start_3;
y_step = y_step_3;
z_up = z_up_3;
z_default = z_default_3;
Serial.println("Mode 3");
if (M != 3)
{
  sit();
  set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_boot);
  set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_boot);
  set_site(2, x_default + x_offset, y_start + y_offset + y_step, z_boot);
  set_site(3, x_default + x_offset, y_start + y_offset + y_step, z_boot);
  wait_all_reach();
  set_site(2, x_default + x_offset, y_start + y_offset, z_boot);
  set_site(3, x_default + x_offset, y_start + y_offset, z_boot);
  wait_all_reach();
  M = 3;
}
switch (data.Mode_3_CMD)
{
  case 0:
    // Serial.println ("Stand");
    stand(data.z_axis);
    break;
  case 1:
    Serial.println("Step forward");
    step_forward_M3(1, data.z_axis, data.Move_speed);
    break;
  case 2:
    Serial.println("Step backward");
    step_backward_M3(1, data.z_axis, data.Move_speed);
    break;
}

```

```

// case 3:
//   Serial.println("Turn left");
//   turn_left_M3(1, data.z_axis, data.Move_speed);
//   break;
// case 4:
//   Serial.println("Turn right");
//   turn_right_M3(1, data.z_axis, data.Move_speed);
//   break;
}
}
else
{
  Serial.print("Stand height");
  Serial.println(data.z_axis);
  stand(data.z_axis);
}
}
void adjust(void)
{
  //initializes eeprom's errors to 0
  //number -100 - +100 is map to 0 - +200 in eeprom
  Serial.println("start adjust");
  for (int i = 0; i < 4; i++)
  {
    for (int j = 0; j < 3; j++)
    {
      EEPROM.write(i * 6 + j * 2, 100);    //write 100 to fit the calculation in polar
to servo
      EEPROM.write(i * 6 + j * 2 + 1, 100); //write 100 to fit the calculation in polar
to servo
    }
  }
  //initializes the relevant variables to adjustment position

```

```

for (int i = 0; i < 4; i++)
{
    set_site(i, adjust_site[0], adjust_site[1], adjust_site[2] + z_absolute);
    for (int j = 0; j < 3; j++)
    {
        site_now[i][j] = site_expect[i][j];
    }
}
//start servo service
FlexiTimer2::set(20, servo_wifi_service);
FlexiTimer2::start();
//initialize servos
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 3; j++)
    {
        servo[i][j].attach(servo_pin[i][j]);
        delay(100);
    }
}
void verify(void)
{
    //calculate correct degree
    float alpha0, beta0, gamma0;
    cartesian_to_polar(alpha0, beta0, gamma0, adjust_site[0], adjust_site[1],
adjust_site[2] + z_absolute);
    //calculate real degree and errors
    float alpha, beta, gamma;
    float degree_error[4][3];
    for (int i = 0; i < 4; i++)
    {

```

```

    cartesian_to_polar(alpha, beta, gamma, real_site[i][0], real_site[i][1],
real_site[i][2] + z_absolute);
    degree_error[i][0] = alpha0 - alpha;
    degree_error[i][1] = beta0 - beta;
    degree_error[i][2] = gamma0 - gamma;
}
//save errors to eeprom
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 3; j++)
    {
        EEPROM.write(i * 6 + j * 2, (int)degree_error[i][j] + 100);
        EEPROM.write(i * 6 + j * 2 + 1, (int)(degree_error[i][j] * 100) % 100 + 100);
    }
}
//initializes the relevant variables to adjustment position
for (int i = 0; i < 4; i++)
{
    set_site(i, adjust_site[0], adjust_site[1], adjust_site[2] + z_absolute);
    for (int j = 0; j < 3; j++)
    {
        site_now[i][j] = site_expect[i][j];
    }
}
//start servo service
FlexiTimer2::set(20, servo_wifi_service);
FlexiTimer2::start();
//initialize servos
for (int i = 0; i < 4; i++)
{
    for (int j = 0; j < 3; j++)
    {
        servo[i][j].attach(servo_pin[i][j]);
    }
}

```

```

        delay(100);
    }
}

void servo_wifi_service(void)
{
    sei();
    if (radio.available())
    {
        radio.read(&data, sizeof(data));
    }
#ifdef SERIAL_PRINT
    Serial.println("adjusted angle");
#endif
    static float alpha, beta, gamma;
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (abs(site_now[i][j] - site_expect[i][j]) >= abs(temp_speed[i][j]))
                site_now[i][j] += temp_speed[i][j];
            else
                site_now[i][j] = site_expect[i][j];
        }
#ifdef SERIAL_PRINT
        Serial.print("Leg:");
        Serial.println(i);
#endif
        cartesian_to_polar(alpha, beta, gamma, site_now[i][0], site_now[i][1],
site_now[i][2]);
        polar_to_servo(i, alpha, beta, gamma);
    }
    // rest_counter++;
}

```



```

}
/*
- set one of end points' expect site
- this founction will set temp_speed[4][3] at same time
- non - blocking function
-----*/
void set_site(int leg, float x, float y, float z)
{
float length_x = 0, length_y = 0, length_z = 0;
if (x != KEEP)
length_x = x - site_now[leg][0];
if (y != KEEP)
length_y = y - site_now[leg][1];
if (z != KEEP)
length_z = z - site_now[leg][2];
float length = sqrt(pow(length_x, 2) + pow(length_y, 2) + pow(length_z, 2));
temp_speed[leg][0] = length_x / length * move_speed * speed_multiple;
temp_speed[leg][1] = length_y / length * move_speed * speed_multiple;
temp_speed[leg][2] = length_z / length * move_speed * speed_multiple;
if (x != KEEP)
site_expect[leg][0] = x;
if (y != KEEP)
site_expect[leg][1] = y;
if (z != KEEP)
site_expect[leg][2] = z;
}
void cartesian_to_polar(volatile float &alpha, volatile float &beta, volatile float
&gamma, volatile float x, volatile float y, volatile float z)
{
//calculate w-z degree
// Serial.print(length_a);
// Serial.print(",");
// Serial.print(length_b);

```

```

// Serial.print(",");
// Serial.println(length_c);
float v, w, h;
float theta, delta;
h = (x >= 0 ? 1 : -1) * (sqrt(pow(x, 2) + pow(y, 2)));
theta = asin(30 / h);
w = h * cos(theta);
v = w - length_c;
alpha = atan2(z, v) + acos((pow(length_a, 2) - pow(length_b, 2) + pow(v, 2) +
pow(z, 2)) / 2 / length_a / sqrt(pow(v, 2) + pow(z, 2)));
beta = acos((pow(length_a, 2) + pow(length_b, 2) - pow(v, 2) - pow(z, 2)) / 2 /
length_a / length_b);
//calculate x-y-z degree
gamma = (w >= 0) ? atan2(y, x) - theta : atan2(-y, -x) + theta;
//trans degree pi->180
alpha = alpha / pi * 180;
beta = beta / pi * 180;
gamma = gamma / pi * 180;
theta = theta / pi * 180;
#ifdef SERIAL_PRINT
Serial.println("expected angle");
Serial.print(alpha);
Serial.print(",");
Serial.print(beta);
Serial.print(",");
Serial.print(gamma);
Serial.print(",");
Serial.println(theta);
delay (1000);
#endif
}
void polar_to_servo(int leg, float alpha, float beta, float gamma)
{

```

```

float alpha_error = EEPROM.read(leg * 6 + 0) - 100 + ((float)EEPROM.read(leg *
6 + 1) - 100) / 100;
float beta_error = EEPROM.read(leg * 6 + 2) - 100 + ((float)EEPROM.read(leg *
6 + 3) - 100) / 100;
float gamma_error = EEPROM.read(leg * 6 + 4) - 100 + ((float)EEPROM.read(leg
* 6 + 5) - 100) / 100;
#ifdef SERIAL_PRINT
Serial.println("Read eeprom memory");
Serial.print("leg:");
Serial.println(leg);
Serial.print(alpha_error);
Serial.print(",");
Serial.print(beta_error);
Serial.print(",");
Serial.println(gamma_error);
#endif
alpha += alpha_error;
beta += beta_error;
gamma += gamma_error;
if (leg == 0)
{
alpha = 90 - alpha;
beta = beta;
gamma = 180 - gamma;
// gamma = (gamma >= 180 ? 180 : gamma);
}
else if (leg == 1)
{
alpha = alpha + 90;
beta = 180 - beta;
gamma = gamma;
// gamma = (gamma <= 0 ? 0 : gamma);
}

```

```

else if (leg == 2)
{
    alpha = 90 - alpha;
    beta = beta;
    gamma = 180 - gamma;
    // gamma = (gamma >= 180 ? 180 : gamma);
}
else if (leg == 3)
{
    alpha = alpha + 90;
    beta = 180 - beta;
    gamma = gamma;
    // gamma = (gamma <= 0 ? 0 : gamma);
}
#ifdef SERIAL_PRINT
Serial.print(alpha);
Serial.print(",");
Serial.print(beta);
Serial.print(",");
Serial.println(gamma);
delay (1000);
#endif
servo[leg][0].write(alpha);
servo[leg][1].write(beta);
servo[leg][2].write(gamma);
}
void wait_reach(int leg)
{
    while (1)
        if (site_now[leg][0] == site_expect[leg][0])
            if (site_now[leg][1] == site_expect[leg][1])
                if (site_now[leg][2] == site_expect[leg][2])
                    break;
}

```

```

}
/*
- wait one of end points move to one site
- blocking function
-----*/
void wait_reach(int leg, float x, float y, float z)
{
while (1)
    if (site_now[leg][0] == x)
        if (site_now[leg][1] == y)
            if (site_now[leg][2] == z)
                break;
}
/*
- wait all of end points move to expect site
- blocking function
-----*/
void wait_all_reach(void)
{
for (int i = 0; i < 4; i++)
    wait_reach(i);
}
/*
- sit
- blocking function
-----*/
void sit(void)
{
move_speed = stand_seat_speed;
for (int leg = 0; leg < 4; leg++)
{
set_site(leg, KEEP, KEEP, z_boot);
}
}

```

```

wait_all_reach();
}
/*
- stand
- blocking function
-----*/
void stand(int z_val)
{
move_speed = stand_seat_speed;
for (int leg = 0; leg < 4; leg++)
{
set_site(leg, KEEP, KEEP, z_default - z_val);
}
wait_all_reach();
}
void step_forward(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_move_speed + M_speed;
while (step-- > 0)
{
if (site_now[3][1] == y_start + y_offset)
{
Serial.println ("Part 1");
//leg 3&1 move
set_site(3, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);
wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);

```

```

set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();

move_speed = leg_move_speed + M_speed;
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();

set_site(1, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();

set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
else
{
Serial.println ("part 2");
//leg 0&2 move
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

wait_all_reach();

move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();

move_speed = leg_move_speed + M_speed;
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);

```

```

wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void step_back(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_move_speed + M_speed;
while (step-- > 0)
{
if (site_now[2][1] == y_start + y_offset)
{
Serial.println ("Part 1");
//leg 2&0 move
set_site(2, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
move_speed = leg_move_speed + M_speed;
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);

```



```

wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
else
{
Serial.println ("Part 2");
//leg 1&3 move
set_site(1, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);
wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

wait_all_reach();
move_speed = leg_move_speed + M_speed;
set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}

```

```

}
void turn_left(unsigned int step, int z_val, int M_speed)
{
    move_speed = spot_turn_speed + M_speed / 2;
    while (step-- > 0)
    {
        if (site_now[2][1] == y_start + y_offset)
        {
            //leg 2&1 move
            set_site(2, x_default + x_offset, y_start + y_offset, z_up);
            wait_all_reach();
            set_site(0, turn_x1 - x_offset, turn_y1, z_default - z_val);
            set_site(1, turn_x0 - x_offset, turn_y0, z_default - z_val);
            set_site(2, turn_x0 + x_offset, turn_y0, z_up);
            set_site(3, turn_x1 + x_offset, turn_y1, z_default - z_val);
            wait_all_reach();
            set_site(2, turn_x0 + x_offset, turn_y0, z_default - z_val);
            wait_all_reach();
            set_site(0, turn_x1 + x_offset, turn_y1, z_default - z_val);
            set_site(1, turn_x0 + x_offset, turn_y0, z_default - z_val);
            set_site(2, turn_x0 - x_offset, turn_y0, z_default - z_val);
            set_site(3, turn_x1 - x_offset, turn_y1, z_default - z_val);
            wait_all_reach();
            set_site(1, turn_x0 + x_offset, turn_y0, z_up);
            wait_all_reach();
            set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
            set_site(1, x_default + x_offset, y_start + y_offset, z_up);
            set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
            set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
            wait_all_reach();
            set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
            wait_all_reach();
        }
    }
}

```

```

else
{
//leg 0&3 move
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_up);
set_site(1, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 - x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(1, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(3, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void turn_right(unsigned int step, int z_val, int M_speed)
{
move_speed = spot_turn_speed + M_speed / 2;
while (step-- > 0)

```

```

{
if (site_now[3][1] == y_start + y_offset)
{
//leg 3&0 move
set_site(3, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(1, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(3, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_default - z_val);
set_site(1, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 - x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
else
{
//leg 1&2 move
set_site(1, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();

```

```

set_site(0, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(1, turn_x0 + x_offset, turn_y0, z_up);
set_site(2, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(3, turn_x1 - x_offset, turn_y1, z_default - z_val);
wait_all_reach();
set_site(1, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(1, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(2, turn_x0 + x_offset, turn_y0, z_default - z_val);
set_site(3, turn_x1 + x_offset, turn_y1, z_default - z_val);
wait_all_reach();
set_site(2, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_up);
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void step_forward_M2(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_trot_speed + M_speed;
while (step-- > 0)
{
if (site_now[3][1] == y_start + y_offset)
{
Serial.println ("Part 1");
//leg 3&1 move

```

```

set_site(3, x_default + x_offset, y_start + y_offset + y_step, z_up - 10);
delay(100);
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(1, x_default + x_offset, y_start + y_offset, z_up + 10);
set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default + x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
}
else
{
Serial.println ("part 2");
//leg 0&2 move
set_site(0, x_default + x_offset, y_start + y_offset + y_step, z_up + 30);
delay(150);
set_site(1, x_default + x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default - x_offset, y_start + y_offset, z_up + 20);
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void step_backward_M2(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_trot_speed + M_speed;
while (step-- > 0)
{
if (site_now[2][1] == y_start + y_offset)
{

```

```

Serial.println ("Part 1");
//leg 2&0 move
set_site(2, x_default + x_offset, y_start + y_offset + y_step, z_up + 20);
delay(100);
set_site(0, x_default + x_offset, y_start + y_offset, z_up + 20);
set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
}
else
{
Serial.println ("Part 2");
//leg 1&3 move
set_site(1, x_default + x_offset, y_start + y_offset + y_step, z_up);
delay(100);
set_site(0, x_default + x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void turn_left_M2(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_trot_speed + M_speed / 2;
while (step-- > 0)
{

```

```

if (site_now[2][1] == y_start + y_offset)
{
    //leg 2&1 move
    set_site(2, x_default + x_offset, y_start + y_offset, z_up);
    wait_all_reach();
    set_site(0, turn_x1 - x_offset, turn_y1, z_default - z_val);
    set_site(1, turn_x0 - x_offset, turn_y0, z_default - z_val);
    set_site(2, turn_x0 + x_offset, turn_y0, z_up);
    set_site(3, turn_x1 + x_offset, turn_y1, z_default - z_val);
    wait_all_reach();
    set_site(2, turn_x0 + x_offset, turn_y0, z_default - z_val);
    wait_all_reach();
    set_site(0, turn_x1 + x_offset, turn_y1, z_default - z_val);
    set_site(1, turn_x0 + x_offset, turn_y0, z_default - z_val);
    set_site(2, turn_x0 - x_offset, turn_y0, z_default - z_val);
    set_site(3, turn_x1 - x_offset, turn_y1, z_default - z_val);
    wait_all_reach();
    set_site(1, turn_x0 + x_offset, turn_y0, z_up);
    wait_all_reach();
    set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
    set_site(1, x_default + x_offset, y_start + y_offset, z_up);
    set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
    set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
    wait_all_reach();
    set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
    wait_all_reach();
}
else
{
    //leg 0&3 move
    set_site(0, x_default + x_offset, y_start + y_offset, z_up);
    wait_all_reach();
    set_site(0, turn_x0 + x_offset, turn_y0, z_up);

```



```

set_site(1, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 - x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(1, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(3, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void turn_right_M2(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_trot_speed + M_speed / 2;
while (step-- > 0)
{
if (site_now[3][1] == y_start + y_offset)
{
//leg 3&0 move
set_site(3, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
}
}
}

```

```

set_site(0, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(1, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(3, turn_x0 + x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_default - z_val);
set_site(1, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(2, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(3, turn_x0 - x_offset, turn_y0, z_default - z_val);
wait_all_reach();
set_site(0, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
else
{
//leg 1&2 move
set_site(1, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, turn_x1 + x_offset, turn_y1, z_default - z_val);
set_site(1, turn_x0 + x_offset, turn_y0, z_up);
set_site(2, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(3, turn_x1 - x_offset, turn_y1, z_default - z_val);
wait_all_reach();
set_site(1, turn_x0 + x_offset, turn_y0, z_default - z_val);

```

```

wait_all_reach();
set_site(0, turn_x1 - x_offset, turn_y1, z_default - z_val);
set_site(1, turn_x0 - x_offset, turn_y0, z_default - z_val);
set_site(2, turn_x0 + x_offset, turn_y0, z_default - z_val);
set_site(3, turn_x1 + x_offset, turn_y1, z_default - z_val);
wait_all_reach();
set_site(2, turn_x0 + x_offset, turn_y0, z_up);
wait_all_reach();
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(2, x_default + x_offset, y_start + y_offset, z_up);
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void step_forward_M3(unsigned int step, int z_val, int M_speed)
{
    move_speed = leg_move_speed + M_speed;
    while (step-- > 0)
    {
        if (site_now[3][1] == y_start + y_offset)
        {
            Serial.println ("Part 1");
            //leg 3&1 move
            set_site(3, x_default + x_offset, y_start + y_offset, z_up);
            wait_all_reach();
            set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
            wait_all_reach();
            set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

```

```

wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default - z_val
- 10);

set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
move_speed = leg_move_speed + M_speed;
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val - 10);
wait_all_reach();
}
else
{
Serial.println ("part 2");
//leg 0&2 move
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val -
10);

set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);

```

```

set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
move_speed = leg_move_speed + M_speed;
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
}
}
void step_backward_M3(unsigned int step, int z_val, int M_speed)
{
move_speed = leg_move_speed + M_speed;
while (step-- > 0)
{
if (site_now[2][1] == y_start + y_offset)
{
Serial.println ("Part 1");
//leg 2&0 move
set_site(2, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(2, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);
wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);
set_site(1, x_default + x_offset, y_start + y_offset, z_default - z_val - 10);
set_site(2, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);

```

```

set_site(3, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
wait_all_reach();
move_speed = leg_move_speed + M_speed;
set_site(0, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(0, x_default + x_offset, y_start + y_offset, z_default - z_val);
wait_all_reach();
}
else
{
Serial.println ("Part 2");
//leg 1&3 move
set_site(1, x_default + x_offset, y_start + y_offset, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(1, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default - z_val
- 10);
wait_all_reach();
move_speed = body_move_speed + M_speed / 2;
set_site(0, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val);
set_site(1, x_default - x_offset, y_start + y_offset + y_step, z_default - z_val -
10);
set_site(2, x_default + x_offset, y_start + y_offset, z_default - z_val);
set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_default -
z_val);
wait_all_reach();
move_speed = leg_move_speed + M_speed;
set_site(3, x_default + x_offset, y_start + y_offset + 2 * y_step, z_up);
wait_all_reach();
set_site(3, x_default + x_offset, y_start + y_offset, z_up);

```

```
wait_all_reach();  
set_site(3, x_default + x_offset, y_start + y_offset, z_default - z_val);  
wait_all_reach();
```

