London South Bank University

Quadrupedal Robotic Platform For Research on Legged Motion Planning

Electronic and Electrical Engineering

Jerome Alexander Graves

Supervised by

Dr. Zhangfang Zhao
School of Engineering

2 May 2020

This report has been submitted for assessment towards a Bachelor of Engineering Degree in electronic and electrical engineering in the School of Engineering, London South Bank University. The report is written in the author's own words and all sources have been properly cited.

Author's signature:

Contents

Acronyms

IK Inverse Kinematics

dof Degrees of Freedom

ROS Robotic Operating System

IMU Inertial Measurment unit

GUI Graphical User Interface

SBC Single Board Computer

CAD Computer-aided Design

UART Universal Asynchronous Receiver/Transmitter

I2C Inter-Integrated Circuit

URDF Universal Robot Description File

Abstract

The goal of this project is the design and creation of a quadrupedal robotic platform and implementation of robotic control algorithms for legged motion planning using ROS.

This project investigates the capability of a designed quadrupedal robot to traverse uneven terrain using control software designed with python and ROS. In a world where over half the earth is inaccessible to wheel robots, the need for legged robots is needed for full automation.

The system allows for basic traversal of terrain by modifying a simple gait algorithm in real-time using an IMU sensor data. This report covers mechanical and electrical design of the robot, as well as the overall software design. The capabilities of the robot are tested and recorded in both flat and uneven terrain by comparing walk speeds and optimal tuning was recommended.

Overall the robot performed adequately but not as well as expected. It is recommended that the control system be improved by adding sensors to the feet of the robot to measure reactionary force on the foot and using more Klamen filtering for the IMU data to stop drift. Further investigation is recommended.

1 Introduction

Over 50 percent of the earth's landmass is inaccessible to wheeled vehicles. This is not the case for a legged creature that can freely traverse most of this geography. This creates a need for legged robot development, with the goal of recreating animals natural ability to traverse uneven terrain[1].

Walking robots have many advantages over traditional wheeled robots but their use in industry and services is still limited [2]. Therefore, they are more suitable to perform tasks in environments that are sensitive to intrusion. A walking robot can benefit from that active suspension as part of its structure and allows the robot to adapt to the terrain. This allows for a smoother ride for any passengers or cargo, or an active omnidirectional but stable base while tasks are being performed with a manipulator. This gives them an advantage in traversing through tight environments. This does depend on its design and control system [3].

The main objective of this project was to test motion planning algorithms with a custom quadrupedal robot. The objectives include the design of a mobile quadrupedal robot with 3 dof Limbs.The specifications inculde battery power, wirelees conectivity, linux compatible SBC, 3dof limbs and IMU feedback. Autodesk fusion360 was used as the mechanical design sofware and 3d printed ABS and lazer cut Ply was used for the manufacturing.

The control system performs a simplified mammalian trot gait by applying out of phase oscillation patterns to the foot positions of the robot. By using the geometry of the leg and an IK solver, necessary joint rotations can be obtain in real-time. To allow the robot to traverse uneven terrain IMU data was used as feedback, with the goal og the system of keeping the body of the robot as level as posible to increase stablility. The control sytstem used ROS' node structure and be implemented using Python. The software comunicates with the robot through wireless conection and inclued a GUI that allows the user to change gait parameters and IMU gains from a remote PC.

The robot and algorythm was tested by timing the robots traversal across 3

types of terrain: Flat, Sloped and with shallow obstiles. The system was tuned by modifying parametres of the gait control system and IMU feedback gain to maximies traversal speed and stability.

2 Aim

The aim of this project was test legged motion control algorithms on a custom designed quadrupedal robot.The robot spesicfications was; 3 dof limbs, IMU, battery powered, wifi enabled SBC. The robot was to be designed in CAD and fabricated using 3D printing and laser cutting techniques. The control system modifies a predefined gait to allow the robot to traverse uneven terrain. The system operates with a ROS master-slave architecture using rViz Visualization software and custom software controller on a remote pc to operate the robot. The robot will be tested on 3 types of terrain, flat, sloped and flat with shallow obstacles. The system will be tuned by modifying the PID gains of the feedback IMU data.

3 Objectives

Design and construction of a quadrupedal robot, with the following specification:

- The robot will be designed using fusion 360 CAD sofware.
- The robot will consist of 4 limbs each with 3 d.o.f. Each limb consists of 3 servos arranged in identically.
- Onboard SBC compatable with linux and ROX with the capability of UART and I2c, battery operation, WiFi-enabled.
- Onboard IMU connected to the SBC via I2C and Powered by a 5 V supply.
- Onboard Servo Controler connected to the SBC via UART and powered by both 7.2 V and 5 V.
- Battery Power and both 5 V and 7.2 V output.

Design and Implementation of GUI using Rviz.

- Configuration of ROS master-slave network. A PC ,with ROS and linux installed, acts as the slave maching and the SBC onboard the robot is the master.
- Configure R-viz simulation of the robot. Done by creation of a URDF file describing the robot geometry and joint motion. This is loaded into R-viz simulation tool. The simulation should broadcast joint positions for the robot to implement.

Configure the robot for use with ROS.

- Creation of a ROS node that implements motions simulated motions from Rviz. This is Run on the robots SBC.
- Creation of a ROS node that sends positional data from the IMU to the motion planning control system. Aso run on the SBC.

Implementation of the motion control system:

- Implementation of an algorithm that calculates joint angles from foot position (IK). This is implemented using ROS node structure.
- Implementation of a ROS node That takes user data from the GUI and IMU data from the robot and generates and runs a gait motion in Rviz. The GUI should consists of a visualisation of the robot and gait motion. The user is also able to change gait parameteres and IMU PID gains in realtime.

Testing of the system:

- Create of a short course for the robot, simulating 3 types of terrain. Flat, Sloped and with shallow obsticles.
- Tune and record robot speed against different gait parameters and IMU PID gains.

4 Deliverables

Design and fabrication of quadrupedal robot with IMU sensor input and to the specifications defined in the objectives.

Design and implementation of a motion control system softwareusing ros and python that uses IK and IMU sensor data as feedback to generate motions for the robot.

Design and Implmetaion of a GUI software in python using Rviz allowing for user to view and configure the generated gate.

Testing the robot across different terrains and recoding results.

Production of the report of results and understading.

5 The Technical Background and Context

5.1 Overview

Quadrupedal robotic designs have many criteria to be considered such as Big-Dog [1], a robotic platform designed addresses the practical problems of on-board power and rough-terrain controls in order to move toward practical legged vehicles. Although robot complexity is high, The concept of robot stability and controls through rough-terrain are universal to all walking robots

5.2 Gaits and balance and control

The gaits of biological Quadrupeds divided into symmetrical and asymmetrical sequences. with the key variables being the duty factor and forelimb-hindlimb phase[4]. And the functions capable of being mathematically modelled as a network of coupled non-linear oscillators[5].

The Best foothold selected should satisfy the following rules as much as possible; foot slip should be avoided or minimized; Walking speed should be maximized. For two candidate footholds, one lies at the front of the default foothold, and the other lies at the back; the priority of the front foothold should be larger than that of the back one. Also, the distance between the candidate foothold and default foothold should be minimized, according to the law of quadruped bionic movement [5].

The control of the robot can be divided into a gait central pattern generator and a control system with feedback linearization. The gait generator is developed based on optimal inverse kinematics with the use of Trigonometric functions[6].

The objective of this project is to create a control system that can traverse across rough terrain. A control system that uses a posture correction controller with data from an IMU is required to modify the foot trajectory [7]. JQuadRobot [8], has an appropriate design with 3 d.o.f. freedom limbs, but the lack of sensor input results in an open-loop control system that allows traversal across flat terrain.

Legged robotic control systems and can be divided into 3 primary activities[1]:

- Supporting the body with a vertical bouncing motion.
- Controlling the altitude of the body by modifying hio torques during each legs stance phase.
- Placing Feet in key locations each step, using symmetry principals.

5.3 Robotic Control Software

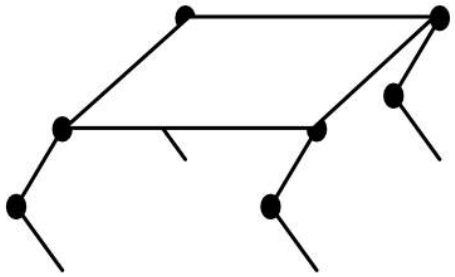
The system uses ROS, a flexible framework for writing robot software. It's purpose is to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. It contains a wide range of libraries that assists with robotic motion and motion planning. Its node structure allows for fast prototyping across multiple machines. The ROS nodes can be implementded with both C++ and Python and are self contained. This allows for exchnaging and testing of elements of the system withough recoding the entire system.

6 Technical Approach

6.1 Mechanical Design

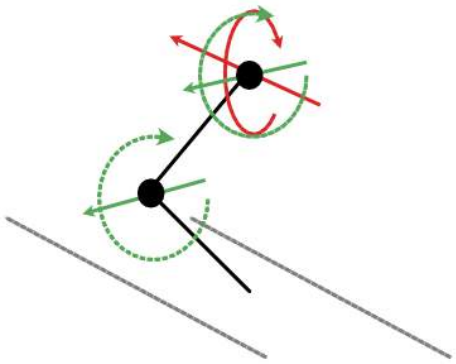
The basic configuration of a quadrupedal robot is a rectangular base uppored by 4 legs in each corner.

Figure 1: Quadruped Reprisentation



There are many leg configurations for quadrupeds. This particular configuration was chosen to emulate mammal gaits with limited d.o.f per limb. Each leg consists of 3 actuators. Two hip actuators which rotate around the X and Y axis and a Knee joint that rotates around the Y axis.

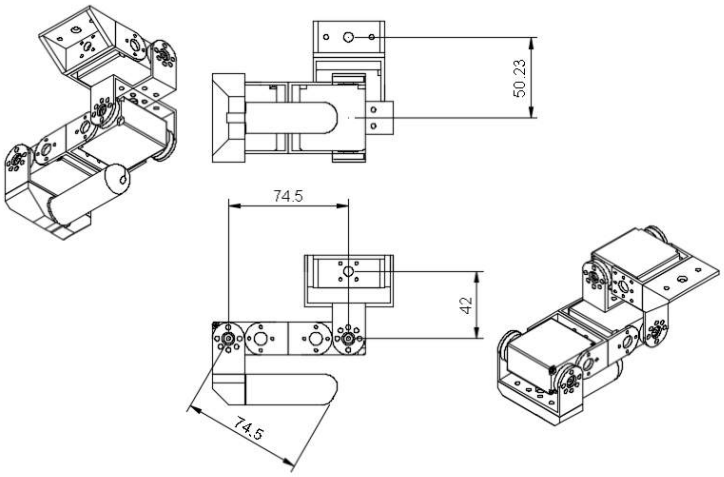
Figure 2: £ degrees of motions Leg



To increase stability limb length was keeped to a minimum. The increased range of motion distance between the hip to knee and knee to foot are keeped same.

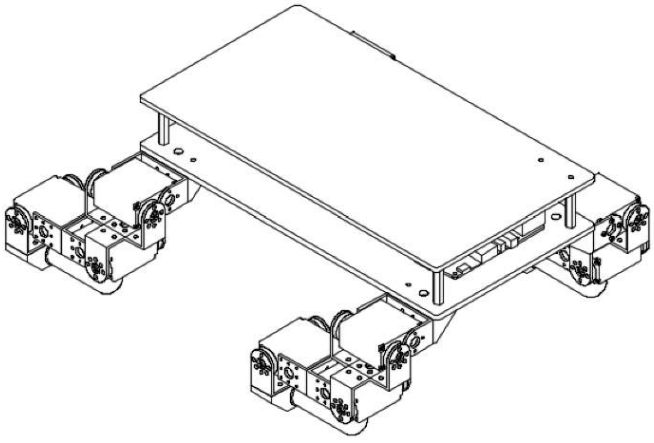
Many leg designs were tested. The figure shows the final design. Lower leg parts and leg mount's were 3D printed from ABS plastic.

Figure 3: Final leg design



A base was made to adequately hold electronics including a battery. This was Laser printed from 4mm ply. Many leg designs were prototyped. Early design

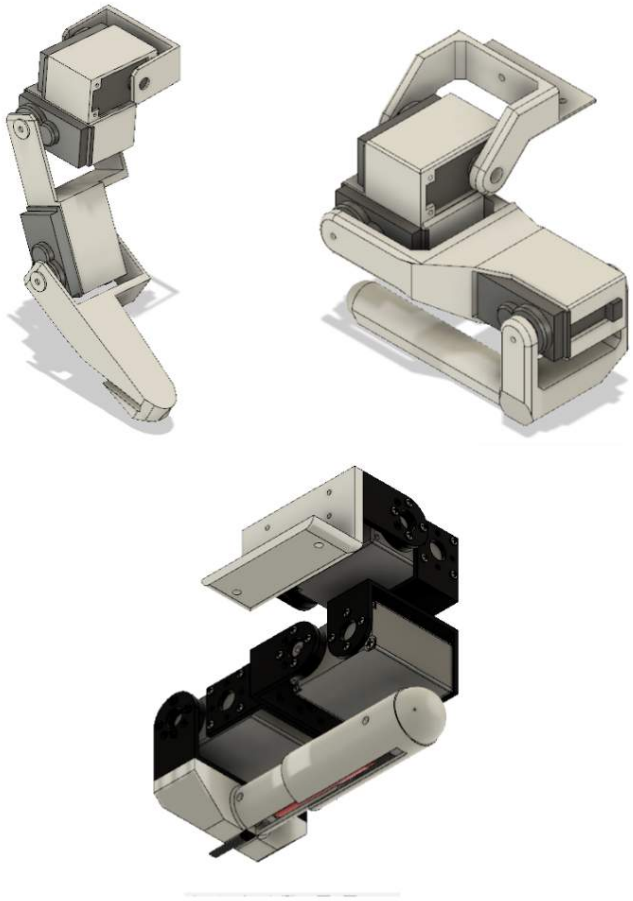
Figure 4: Final robot design



lacked freedom of movement and ability to sit flat without active actuation. Limb length and body size was also a factor, with a shorter limb and wider body

being preferable. This has the advantage of greater stability and energy efficiency but has the disadvantage of decreasing mobility. A fully 3D printed leg was also tested, this was unable to perform under the chaotic moments of the legged motion.

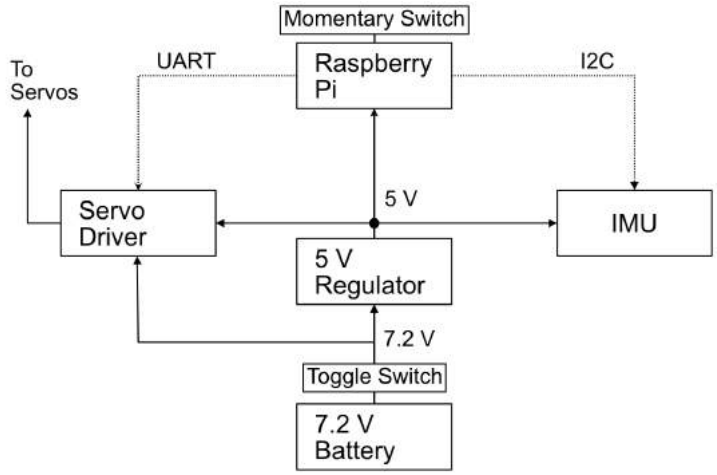
Figure 5: Leg design iterations



6.2 Electronic Design

The basic electrical specification for the robot is the capability to drive power and control 12 servo motors. A single board computer to operate ROS. An IMU with 9 d.o.f for sensory input and a Battery power system. The basic topology of the system is shown in the following figure.

Figure 6: Flow diagram of electronics

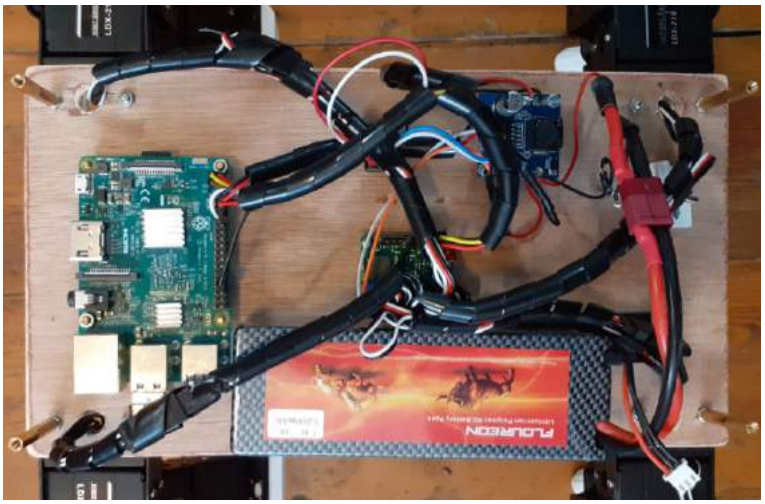


The system will consist of 2 voltages 7.2 V for the servo and 5 V for the single board computer. This will be supplied with a 2 S Lithium polymer battery with an average voltage of 7.4V. A voltage regulator is used to step down the voltage to 5 V. There are 2 external switches connected to the system. The Momentary switch initiates safe shutdown and disables servo motors and the toggle switch disconnects the battery from the system.

The single board computer uses is a Raspberry Pi 3B+ with Lubuntu installed.

The IMU (ICM20948 9DoF Motion sensor) is connected to the 5v supply and communicates with the Raspberry Pi with I2C connection. The Servo controller board (Mini Maestro 12-Channel) is connected to the Raspberry Pi via UART and requires both 7.4 V and 5 V. The servos are controllers with a PWM signal with a max voltage of 5V generated by the servo controller. The Implemented electronics is shown in the figure.

Figure 7: Robot electronics

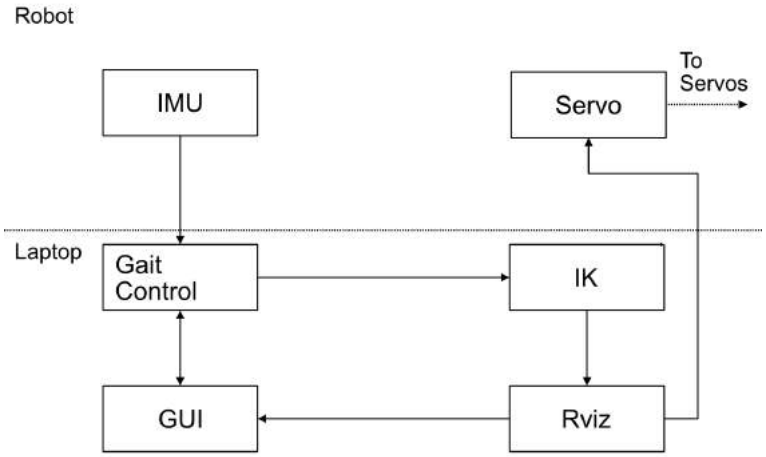


The issue that was come across in the electrically design was that the servo controller could not supply enough current to the servos and had to be replaced with a controller with a higher power rating.

6.3 Software design

The base specification of the software is to link to the robot via wifi and control its actuators depending on a generated gait sequence and data from the IMU on the robot. The software was designed using python and ROS' node structure. Node communicated across a LAN using channels with unique identifiers called topics. Topics are configured to receive data in a certain format. The basic topology is shown in the figure. The system is designed so that the end user

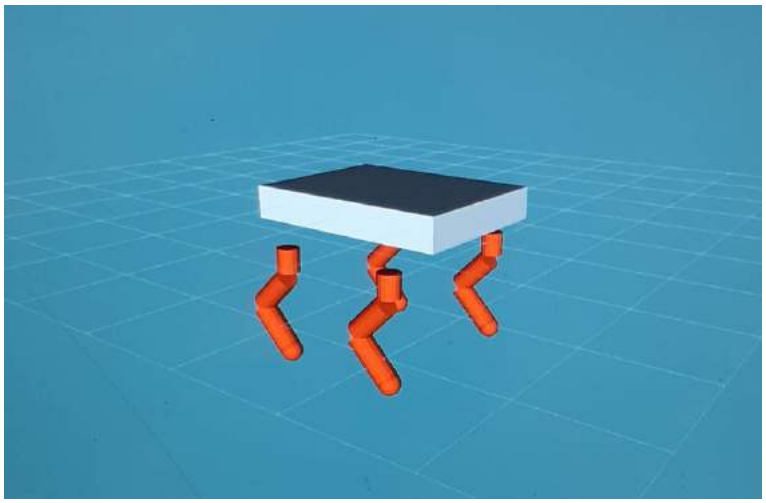
Figure 8: Flow diagram ROS nodes



can simulate motions before applying them with the robot. This can be done by disabling the servo node. The servo node's purpose is to apply joint rotation data to the servos through the servo controller. It Receives joint state data on the joint topic generated by Rviz which is an array of joint rotations in radians. This data is converted into a PWM signal and sent to the servo controller via UART. IMU node Receives rotational data from the IMU via I2C connection and publishes the data on the IMU topic. The data received from the IMU is very noisy as such the data is passed through a low pass filter to stabilise the information. Over damping of the data results in a slower response to rotational changes. This is a subject to be further explored. The IK node receives the planned position of the end of the legs and converts this into joint rotations to be published on the joint topic. It uses the Ik topic to receive data in the format of an array of 4 points, with each point consisting of 3 32bit floats

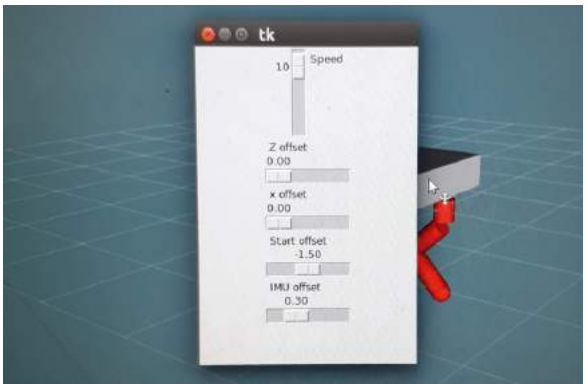
representing x,y,z. It Sends data on the joint topic in the format of an array of 12 64 bit floats representing the angle of each joint in radians. It uses the ikpy library to calculate the inverse kinematics using the geometry and limits of a leg. Due to the Heavy overhead of the IK calculation the number of iteration of calculations are limited to 10. Reducing iterations increases accuracy of the calculation but increases the speed and therefore the response of the robot to external forces. This is to be further explored. Rviz is a visualization software supplied with ROS. It was configured to display robot motion. A URDF file was created to allow Rviz to display a 3D model of the robot. Rviz then listens for messages on the joints topic and translates limbs accordingly.The message format is an array of 12 64 bit floats. The Gait control node is the main controller for the ro-

Figure 9: Rviz Robot Simulation



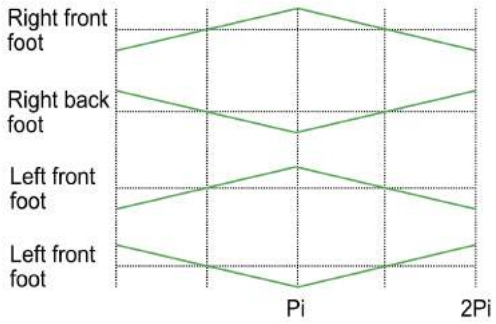
bot. It receives rotational data from the IMU topic user input from a GUI and generates a stream of foot positions using sinusoidal motion. The user parameters are speed of walk; X and Z offset, which control how far forward and high the leg extends during a step. Start offset sets how high the body of the robot remains off the ground. The P, I and D IMU offset adds or reduces the PID gain of the IMU data.

Figure 10: GUI



The gait template is of a simple mammalian walk in a forward direction. Each diagonal limb acting in unison and each adjacent leg having an identical sequence 180 degree out of phase. Each leg sequence consists of 2 phases. The foot moves forward in an ark motion, lifting the foot vertically. Then dragging the foot across the ground backwards.

(a) X axis motion



(b) Y axis motion

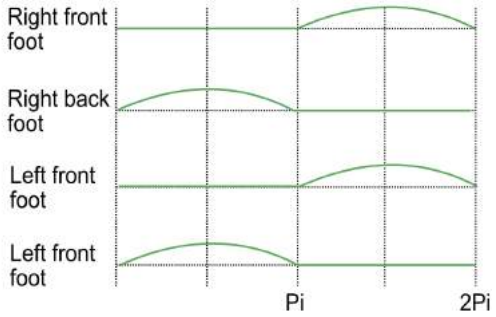


Figure 11: Y and X motion of feet

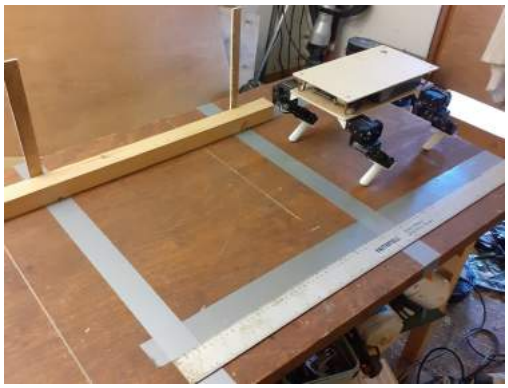
7 Results And Disscusion

The algorithm was tested on multiple surfaces. The first is a flat hard surface. A surface literied with short obstacles secured to the ground. And the last is terrain with ramps.

First the robot was tested on flat terrain with IMU PID gains set to 0. The parameters were then adjusted to create as stable a walk as possible before adding input from the IMU input. The speed of the robot was calculated by timing with the robot across a 50cm distance. The highest speed traversal was considered the most stable tuning.

Without IMU input the walk is very unstable. With this gait there is a point where the robot has to balance on 2 legs. Without input the robot struggles due

(a) Flat



(b) Sloped

(c) Obstacles

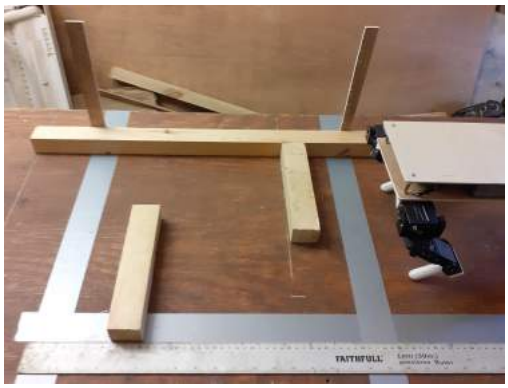


Figure 12: Types of terrain

to its center of gravity being off center. This can be compensated against by reducing set size and increasing the speed of the steps. But too much speed can also cause instability.

Table 1: Optimal Flat terrain tuning with no IMU gain

Step Speed(sec)	0.73
Vertial Leg travel(mm)	15
Horizontal Leg travel(mm)	16
Vertial Body offset(mm)	1.43

The speed of the robot was then tested on flat terrain modifying IMU PID input.

Table 2: Optimal Flat terrain tuning with IMU gain

IMU P gain	0.1
IMU I gain	0
IMU D gain	0.1
Liniar Speed(mm/s)	41.7

The speed of the robot was then tested on flat terrain littered with set obstacles modifying IMU PID input.

Table 3: Optimal Obstacles terrain tuning with IMU gain

IMU P gain	0.3
IMU I gain	0
IMU D gain	0.1
Liniar Speed(mm/s)	4.4

The speed of the robot was then tested on sloped terrain modifying IMU PID input.

Table 4: Optimal sloped terrain tuning with IMU gain

IMU P gain	0.2
IMU I gain	0
IMU D gain	0.3
Liniar Speed(mm/s)	34.5

8 Conclusions and Recommendations for Further Work

The robot is adequate and can complete the task of traversal over uneven terrain but performed worse than expected with a very slow maximum walk speed and is very sensitive to change in values. It is than adequate for a complete system due to limitations in both the software and the hardware.

Computing power is a limitation due to the strain of calculating the Ik of the limbs so many times a second with faster hardware the accuracy and speed of the IK calculation could be improved. It could also be improved by rewriting the IK algorithm in C++ for faster computation. The result would be higher resolution motions and more accurate leg positioning.

Another issue is the response of the servos is slow. Due to this it is difficult for the robot to respond to sudden changes in the environment. It's recommended to use a higher power actuator with a lower gearing. Brushless motors with either planetary or harmonic gearbox is recommended.

The IMU data tends to drift giving inaccurate data over time. It is recommended to improve its accuracy by implementing a kalman filter which estimates the position of a system.

For an Improved system the amount of sensory data the robot receives should be increased. Pressure sensors on the feet should measure the reactionary forces between the robot and the ground plain. This would allow for greater stability [1].

Leg design could be improved. The final design chosen does not have complete range of motion and as a result the legs require a certain level of extention for the robot to walk. In general for further explorations it is recommended to investigate this subject further with higher power actuators with joint position and current feedback. Also test modification of the filtering of IMU data.

9 References

[1] M. Raibert, K. Blankespoor, G. Nelson and R. Playter, 'BigDog, the rough-terrain quadruped robot', *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10 822–10 825, 2008. DOI: 10.3182/20080706-5-KR-1001.01833.

[2] M. Hildebrand, 'Vertebrate locomotion: An introduction: How does an animal's body move itself along?', *BioScience*, vol. 39, no. 11, pp. 764–765, Dec. 1989. DOI: 10.1093/bioscience/39.11.764.

[3] E. Massi, L. Vannucci, U. Albanese, M. C. Capolei, A. Vandesompele, G. Urbain, A. M. Sabatini, J. Dambre, C. Laschi, S. Tolu and E. Falotico, 'Combining evolutionary and adaptive control strategies for quadruped robotic locomotion', *Frontiers in Neurorobotics*, vol. 13, 29 Aug. 2019. DOI: 10.3389/fnbot.2019.00071.

[4] P. González-de-Santos, E. Garcia and J. Estremera, *Quadrupedal locomotion: an introduction to the control of four-legged robots*, 04. American Library Association, Dec. 2006, vol. 44, pp. 44–47.

[5] S. Vatau, V. Ciupe, C. Moldovan and I. Maniu, 'Mechanical design and system control of quadruped robot', in *MECHANIKA*, vol. 5, Jan. 2010.

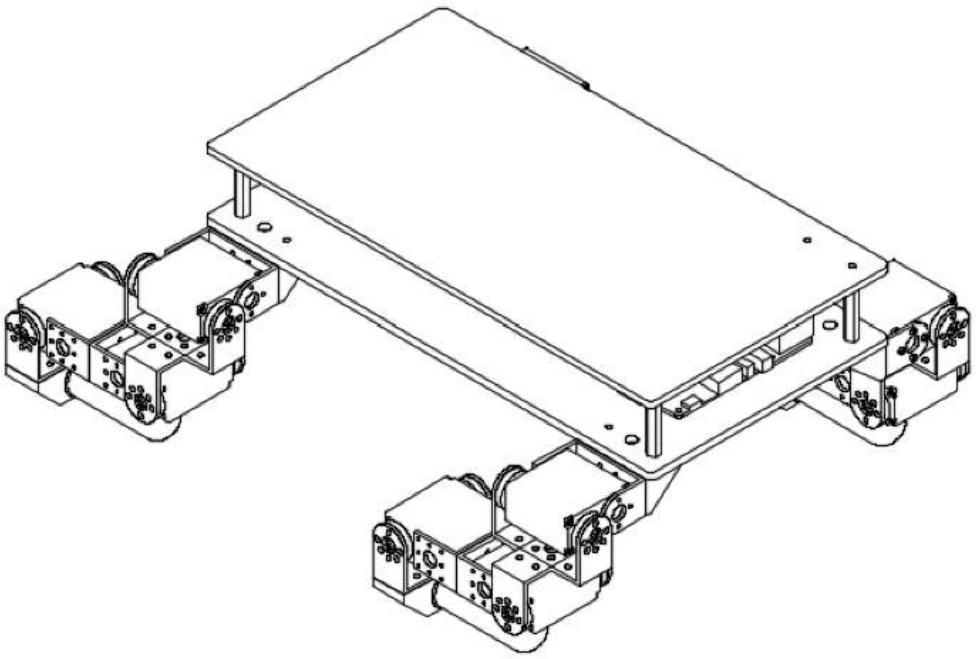
[6] K. Yang, Y. Li, L. Zhou and X. Rong, 'Energy efficient foot trajectory of trot motion for hydraulic quadruped robot', *Energies*, vol. 12, no. 13, p. 2514, Jun. 2019. DOI: 10.3390/en12132514.

[7] K. Tsujita, K. Tsuchiya and A. Onat, 'Adaptive gait pattern control of a quadruped locomotion robot', in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, vol. 4, Maui, HI, USA: IEEE, 2001. DOI: 10.1109/iros.2001.976416.

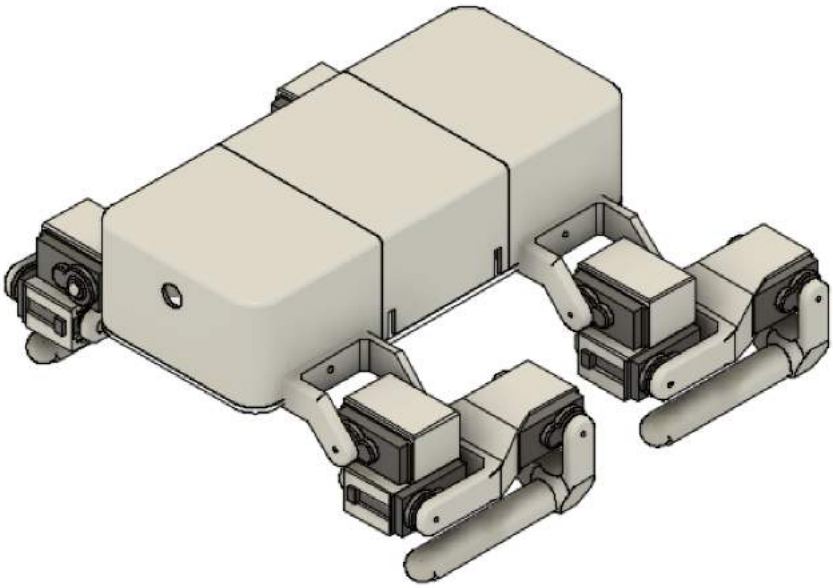
[8] C. Yu, L. Zhou, H. Qian and Y. Xu, 'Posture correction of quadruped robot for adaptive slope walking', in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia: IEEE, Dec. 2018. DOI: 10.1109/robio.2018.8665093.

10 Appendix

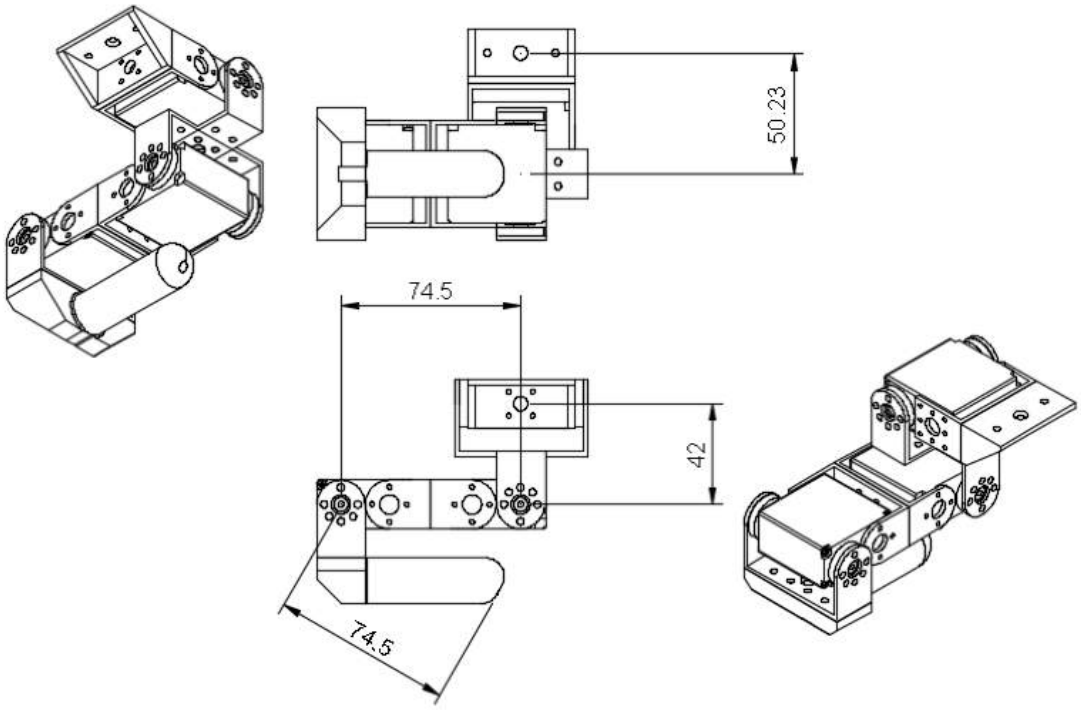
10.1 Robot Final Design



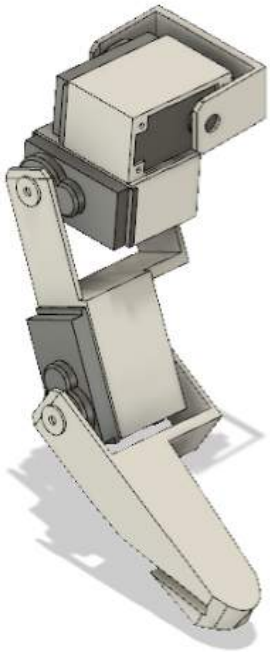
10.2 Previous Robot Design

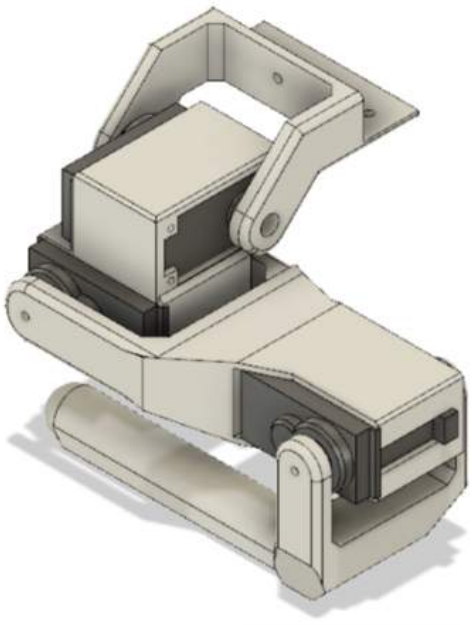


10.3 Leg Design



10.4 Previous Leg Designs





10.5 URDF file

```
<?xml version="1.0" encoding="UTF-8"?>
<robot name="ROSberryPup"><!--Robot name-->
  <!--Geometry, Position and Joint Parameters-->
  <!--Base Link-->
  <link name="base_link">
    <visual>
      <geometry>
        <box size="2.8 2.2 0.43"/><!--Geometry shape and size. -->
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0"/><!--Offset from origin.-->
    </visual>
  </link>

  <!--Right Front Leg-->
  <link name="rf_hip">
    <visual>
      <geometry>
        <cylinder length="0.32" radius="0.15"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 -0.21"/>
    </visual>
  </link>
  <joint name="base_to_rf_hip" type="revolute"><!--Joint name an
   type.-->
    <parent link="base_link"/><!--Joint parent conection.-->
    <child link="rf_hip"/><!--Joint child connection.-->
    <axis xyz="1 0 0" /><!--Joint rotational axis.-->
    <origin rpy="-1.5707 0 0" xyz="-0.83 1.18 -0.455"/><!--Joint
   posiotn.-->
    <limit effort="50" velocity="10" lower="0" upper="2.141" /><!
   --Joint limits and speed.-->
  </joint>

  <link name="rf_thigh">
    <visual>
      <geometry>
        <cylinder length="0.64" radius="0.15"/>
      </geometry>
```

```
36      <origin rpy="0 0 0" xyz="0 0 -0.37"/>
37    </visual>
38  </link>
39  <joint name="rf_hip_to_thigh" type="revolute">
40    <parent link="rf_hip"/>
41    <child link="rf_thigh"/>
42    <axis xyz="0 1 0" />
43    <origin rpy="0 -1.5707 0" xyz="0 0 -0.42"/>
44    <limit effort="50" velocity="10" lower="0" upper="3.141" />
45  </joint>
46
47  <link name="rf_shin">
48    <visual>
49      <geometry>
50        <cylinder length="0.64" radius="0.15"/>
51      </geometry>
52      <origin rpy="0 0 0" xyz="0 0 -0.37"/>
53    </visual>
54  </link>
55  <joint name="rf_thigh_to_shin" type="revolute">
56    <parent link="rf_thigh"/>
57    <child link="rf_shin"/>
58    <axis xyz="0 1 0" />
59    <origin rpy="0 0 0" xyz="0 0 -0.74"/>
60    <limit effort="50" velocity="10" lower="0" upper="3.141" />
61  </joint>
62
63  <link name="rf_foot">
64    <visual>
65      <geometry>
66        <sphere radius="0.15"/>
67      </geometry>
68      <origin rpy="0 0 0" xyz="0 0 0"/>
69    </visual>
70  </link>
71  <joint name="rf_shin_to_foot" type="fixed">
72    <parent link="rf_shin"/>
73    <child link="rf_foot"/>
74    <axis xyz="0 1 0" />
```

```
75      <origin rpy="0 0 0" xyz="0 0 -0.74"/>
76    </joint>
77
78    <!--Right Back Leg-->
79    <link name="rb_hip">
80      <visual>
81        <geometry>
82          <cylinder length="0.32" radius="0.15"/>
83        </geometry>
84        <origin rpy="0 0 0" xyz="0 0 -0.21"/>
85      </visual>
86    </link>
87    <joint name="base_to_rb_hip" type="revolute">
88      <parent link="base_link"/>
89      <child link="rb_hip"/>
90      <axis xyz="1 0 0" />
91      <origin rpy="-1.5707 0 0" xyz="0.83 1.18 -0.455"/>
92      <limit effort="50" velocity="10" lower="0" upper="2.141" />
93    </joint>
94
95    <link name="rb_thigh">
96      <visual>
97        <geometry>
98          <cylinder length="0.64" radius="0.15"/>
99        </geometry>
100       <origin rpy="0 0 0" xyz="0 0 -0.37"/>
101     </visual>
102   </link>
103   <joint name="rb_hip_to_thigh" type="revolute">
104     <parent link="rb_hip"/>
105     <child link="rb_thigh"/>
106     <axis xyz="0 1 0" />
107     <origin rpy="0 -1.5707 0" xyz="0 0 -0.42"/>
108     <limit effort="50" velocity="10" lower="0" upper="3.141" />
109   </joint>
110
111   <link name="rb_shin">
112     <visual>
113       <geometry>
```

```
      <cylinder length="0.64" radius="0.15"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 -0.37"/>
  </visual>
</link>
<joint name="rb_thigh_to_shin" type="revolute">
  <parent link="rb_thigh"/>
  <child link="rb_shin"/>
  <axis xyz="0 1 0" />
  <origin rpy="0 0 0" xyz="0 0 -0.74"/>
  <limit effort="50" velocity="10" lower="0" upper="3.141" />
</joint>

<link name="rb_foot">
  <visual>
    <geometry>
      <sphere radius="0.15"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 0"/>
  </visual>
</link>
<joint name="rb_shin_to_foot" type="fixed">
  <parent link="rb_shin"/>
  <child link="rb_foot"/>
  <axis xyz="0 1 0" />
  <origin rpy="0 0 0" xyz="0 0 -0.74"/>
</joint>

<!--Left Front Leg-->
<link name="lf_hip">
  <visual>
    <geometry>
      <cylinder length="0.32" radius="0.15"/>
    </geometry>
    <origin rpy="0 0 0" xyz="0 0 -0.21"/>
  </visual>
</link>
<joint name="base_to_lf_hip" type="revolute">
  <parent link="base_link"/>
```

```
153     <child link="lf_hip"/>
154     <axis xyz="1 0 0" />
155     <origin rpy="-1.5707 0 0" xyz="0.83 -1.18 -0.455"/>
156     <limit effort="50" velocity="10" lower="0" upper="2.141" />
157   </joint>
158 
159   <link name="lf_thigh">
160     <visual>
161       <geometry>
162         <cylinder length="0.64" radius="0.15"/>
163       </geometry>
164       <origin rpy="0 0 0" xyz="0 0 -0.37"/>
165     </visual>
166   </link>
167   <joint name="lf_hip_to_thigh" type="revolute">
168     <parent link="lf_hip"/>
169     <child link="lf_thigh"/>
170     <axis xyz="0 1 0" />
171     <origin rpy="0 -1.5707 0" xyz="0 0 -0.42"/>
172     <limit effort="50" velocity="10" lower="0" upper="3.141" />
173   </joint>
174 
175   <link name="lf_shin">
176     <visual>
177       <geometry>
178         <cylinder length="0.64" radius="0.15"/>
179       </geometry>
180       <origin rpy="0 0 0" xyz="0 0 -0.37"/>
181     </visual>
182   </link>
183   <joint name="lf_thigh_to_shin" type="revolute">
184     <parent link="lf_thigh"/>
185     <child link="lf_shin"/>
186     <axis xyz="0 1 0" />
187     <origin rpy="0 0 0" xyz="0 0 -0.74"/>
188     <limit effort="50" velocity="10" lower="0" upper="3.141" />
189   </joint>
190 
191   <link name="lf_foot">
```

```
192     <visual>
193       <geometry>
194         <sphere radius="0.15"/>
195       </geometry>
196       <origin rpy="0 0 0" xyz="0 0 0"/>
197     </visual>
198   </link>
199   <joint name="lf_shin_to_foot" type="fixed">
200     <parent link="lf_shin"/>
201     <child link="lf_foot"/>
202     <axis xyz="0 1 0" />
203     <origin rpy="0 0 0" xyz="0 0 -0.74"/>
204   </joint>
205   <!--Left Back Leg-->
206
207   <link name="lb_hip">
208     <visual>
209       <geometry>
210         <cylinder length="0.32" radius="0.15"/>
211       </geometry>
212       <origin rpy="0 0 0" xyz="0 0 -0.21"/>
213     </visual>
214   </link>
215   <joint name="base_to_lb_hip" type="revolute">
216     <parent link="base_link"/>
217     <child link="lb_hip"/>
218     <axis xyz="1 0 0" />
219     <origin rpy="-1.5707 0 0" xyz="-0.83 -1.18 -0.455"/>
220     <limit effort="50" velocity="10" lower="0" upper="2.141" />
221   </joint>
222
223   <link name="lb_thigh">
224     <visual>
225       <geometry>
226         <cylinder length="0.64" radius="0.15"/>
227       </geometry>
228       <origin rpy="0 0 0" xyz="0 0 -0.37"/>
229     </visual>
230   </link>
```

```
231   <joint name="lb_hip_to_thigh" type="revolute">
232     <parent link="lb_hip"/>
233     <child link="lb_thigh"/>
234     <axis xyz="0 1 0" />
235     <origin rpy="0 -1.5707 0" xyz="0 0 -0.42"/>
236     <limit effort="50" velocity="10" lower="0" upper="3.141" />
237   </joint>
238 
239   <link name="lb_shin">
240     <visual>
241       <geometry>
242         <cylinder length="0.64" radius="0.15"/>
243       </geometry>
244       <origin rpy="0 0 0" xyz="0 0 -0.37"/>
245     </visual>
246   </link>
247   <joint name="lb_thigh_to_shin" type="revolute">
248     <parent link="lb_thigh"/>
249     <child link="lb_shin"/>
250     <axis xyz="0 1 0" />
251     <origin rpy="0 0 0" xyz="0 0 -0.74"/>
252     <limit effort="50" velocity="10" lower="0" upper="3.141" />
253   </joint>
254 
255   <link name="lb_foot">
256     <visual>
257       <geometry>
258         <sphere radius="0.15"/>
259       </geometry>
260       <origin rpy="0 0 0" xyz="0 0 0"/>
261     </visual>
262   </link>
263   <joint name="lb_shin_to_foot" type="fixed">
264     <parent link="lb_shin"/>
265     <child link="lb_foot"/>
266     <axis xyz="0 1 0" />
267     <origin rpy="0 0 0" xyz="0 0 -0.74"/>
268   </joint>
269 </robot>
```

10.6 IMU node python code

```
#!/usr/bin/env python
import rospy, math
from geometry_msgs.msg import  Quaternion
from L3GD20 import L3GD20
import time



def talker():
    # Communication object.
    s = L3GD20(busId = 1, slaveAddr = 0x69, ifLog = False,
   ifWriteBlock=False)


    # Preconfiguration
    s.Set_PowerMode("Normal")
    s.Set_FullScale_Value("250dps")
    s.Set_AxisX_Enabled(True)
    s.Set_AxisY_Enabled(True)
    s.Set_AxisZ_Enabled(True)

    # Print current configuration.
    s.Init()
    s.Calibrate()

    # Calculation values.
    dt = 0.1
    x1 = 0
    y1= 0
    z1 = 0
    w1 = 1

    # Start ROS node.
    pub = rospy.Publisher('IMU', Quaternion, queue_size=1)
    rospy.init_node('IMU', anonymous=False)
    rate = rospy.Rate(10) # 10hz

    # Position Calculation
```

```
38      while not rospy.is_shutdown():
39          dxyz = s.Get_CalOut_Value()
40          x1 += dxyz[0]*dt;
41          y1 += dxyz[1]*dt;
42          z1 += dxyz[2]*dt;
43          w1 = 1
44          q = Quaternion(x=math.radians(x1),y=math.radians(y1),z=
    math.radians(z1),w=1)
45          # Send data through ROS.
46          rospy.loginfo(q)
47          pub.publish(q)
48          rate.sleep()
49
50  if __name__ == '__main__':
51      try:
52          talker()
53      except rospy.ROSInterruptException:
54          pass
```

10.7 Servo node python code

```
1  #!/usr/bin/env python
2  import rospy , ServoLib
3  from sensor_msgs.msg import JointState
4
5  # Start communication with sero controller.
6  servo = ServoLib
7  servo.setup()
8
9
10 def callback(msg):
11     # Set servo poistions.
12     joint = msg.position
13     servo.setDirectionRad(0,joint[0],0)
14     servo.setDirectionRad(1,joint[1],1)
15     servo.setDirectionRad(2,joint[2],0)
16
17     servo.setDirectionRad(4,joint[3],0)
18     servo.setDirectionRad(5,joint[4],0)
19     servo.setDirectionRad(6,joint[5],1)
```

```
20
21      servo.setDirectionRad(8,joint[6],0)
22      servo.setDirectionRad(9,joint[7],0)
23      servo.setDirectionRad(10,joint[8],1)
24
25      servo.setDirectionRad(12,joint[9],0)
26      servo.setDirectionRad(13,joint[10],0)
27      servo.setDirectionRad(14,joint[11],1)
28
29  def listener():
30
31      # Start ROS node.
32      rospy.init_node('Servo', anonymous=False)
33      # Listen for messages on joint channel.
34      rospy.Subscriber("joint", JointState, callback)
35      #Keeps python from exiting until this node is stopped.
36      rospy.spin()
37
38  if __name__ == '__main__':
39      print ('start')
40      listener()
```

10.8 IK node python code

```
1  #!/usr/bin/env python
2
3  import rospy
4  from ikpy.chain import Chain
5  from ikpy.link import OriginLink, URDFLink
6
7  from std_msgs.msg import Int32, Header
8  from sensor_msgs.msg import JointState
9  from geometry_msgs.msg import Pose, Polygon, Point32
10 global joint, rf_pos, rb_pos, lf_pos, lb_pos
11
12
13 point = Point32()
14 point.x = 0
15 point.y = 0
16 point.z = -1.6
```

```
points = Polygon()

for i in range(4):
  points.points.append(point)

rf_pos = points.points[0]
rb_pos = points.points[1]
lf_pos = points.points[2]
lb_pos = points.points[3]

joint = JointState()
h = Header()


chain = Chain(name='leg', links=[
    OriginLink(),
    URDFLink(
      name="hip",
      translation_vector=[0, 0, 0],
      orientation=[1.57, 0, 0],
      rotation=[1, 0, 0],
      bounds=[1.57,3.141]
    ),
    URDFLink(
      name="thigh",
      translation_vector=[0, 0, 0.42],
      orientation=[0, 0, 0],
      rotation=[0, 1, 0],
      bounds=[0,3.141]
    ),
    URDFLink(
      name="shin",
      translation_vector=[-0.745, 0, 0],
      orientation=[0, 0, 0],
      rotation=[0, 1, 0],
      bounds=[0,3.141]
    ),
    URDFLink(
      name="foot",
```

```
      translation_vector=[-0.745, 0, 0],
      orientation=[0, 0, 0],
      rotation=[1, 1, 1],
    )
])

def callback(msg):
    global rf_pos, rb_pos, lf_pos, lb_pos
    rf_pos=msg.points[0]
    rb_pos=msg.points[1]
    lf_pos=msg.points[2]
    lb_pos=msg.points[3]
    #print (msg)

def ik(x,y,z):
  ik=chain.inverse_kinematics([
            [1, 0, 0, x],
            [0, 1, 0, y],
            [0, 0, 1, z],
            [0, 0, 0, 1]
            ],max_iter=5)

  return(ik)

if __name__=='__main__':
    rospy.init_node('ik_node')
    sub=rospy.Subscriber('ik', Polygon, callback)
    pub=rospy.Publisher('joint', JointState, queue_size=5)


    rate=rospy.Rate(30)
    while not rospy.is_shutdown():
    rf_ik = ik(rf_pos.x,rf_pos.y,rf_pos.z)
    rb_ik = ik(rb_pos.x,rb_pos.y,rb_pos.z)
    lf_ik = rf_ik#ik(lf_pos.x,lf_pos.y,lf_pos.z)
    lb_ik = rb_ik#ik(lb_pos.x,lb_pos.y,lb_pos.z)

    h.stamp = rospy.Time.now()
    joint.header = h
```

```
95      joint.name = ['base_to_rf_hip', 'rf_hip_to_thigh', '
        rf_thigh_to_shin', 'base_to_rb_hip', 'rb_hip_to_thigh','
        rb_thigh_to_shin', 'base_to_lf_hip', 'lf_hip_to_thigh', '
        lf_thigh_to_shin', 'base_to_lb_hip','lb_hip_to_thigh', '
        lb_thigh_to_shin']
96      joint.position = [rf_ik[1],rf_ik[2],rf_ik[3],
97                        rb_ik[1],rb_ik[2],rb_ik[3],
98                        lf_ik[1],lf_ik[2],lf_ik[3],
99                        lb_ik[1],lb_ik[2],lb_ik[3],]
100     pub.publish(joint)
101     rf_old = rf_ik
102     rb_old = rb_ik
103     lf_old = lf_ik
104     lb_old = rb_ik
105
106     rate.sleep()
```

10.9 Gait Control node python code

```
1  #!/usr/bin/env python
2  import rospy, math, PID # ROS python, PID and Maths Library.
3  from geometry_msgs.msg import Polygon, Point32, Pose, Quaternion #
       ROS messages.
4  from Tkinter import * # GUI Library.
5
6  # After data from  IMU the IMU node arrives update the IMU offset.
7  def callback(msg):
8    global imu_off
9    imu_off = msg
10
11 # main loop
12 def talker():
13   # Initialise IMU variables.
14   global imu_off
15   imu_tf = [0,0,0,0]
16   imu_off = Quaternion()
17
18   #Initialise GUI.
19   top = Tk()
20   #Speed of movment.
```

```
21  spdW = Scale(top,from_=5, to=100, label='Speed')
22  spdW.pack()
23  #Vertial Leg travel.
24  zW = Scale(top,from_=0, to=1, resolution=0.01, label='Z offset',
      orient=HORIZONTAL)
25  zW.pack()
26  #Horizontal Leg travel.
27  xW = Scale(top,from_=0, to=1, resolution=0.01, label='x offset',
      orient=HORIZONTAL)
28  xW.pack()
29  # Vertial Body offset
30  stW = Scale(top,from_=-1, to=-2, resolution=0.01, label='Start
     offset', orient=HORIZONTAL)
31  stW.pack()
32  #IMU P gain.
33  imuP = Scale(top,from_=0, to=1, resolution=0.01, label='IMU P
     Gain', orient=HORIZONTAL)
34  imuP.pack()
35  #IMU I gain.
36  imuI = Scale(top,from_=0, to=1, resolution=0.01, label='IMU I
     gain', orient=HORIZONTAL)
37  imuI.pack()
38  #IMU D gain.
39  imuD = Scale(top,from_=0, to=1, resolution=0.01, label='IMU D
     gain', orient=HORIZONTAL)
40  imuD.pack()
41
42
43  # Connect to ROS IMU and IK node.
44  sub=rospy.Subscriber('IMU', Quaternion, callback)
45  pub = rospy.Publisher('ik',Polygon, queue_size=1)
46  rospy.init_node('Planner', anonymous=False)
47
48
49  # Timing variables
50  rate = rospy.Rate(30) # ROS refresh rate in Hz.
51  tik = 0 # counter 1
52  t_run =30 # counter end
53  tik2 = t_run/2 # counter 2, starts 180 degrees out of phase. .
```

```
54
55  # Set Default GUI values
56  spdW.set(10)
57  zW.set(0)
58  xW.set(0)
59  stW.set(-1.5)
60  imuW.set(0.3)
61
62  #Set up PID controller
63  pidx = PID.PID(inP, inI, inD)
64  pidx.SetPoint = 0
65  pidx.setSampleTime(1)
66
67  pidy = PID.PID(inP, inI, inD)
68  pidy.SetPoint = 0
69  pidy.setSampleTime(1)
70
71  # ROS loop
72  while not rospy.is_shutdown():
73    # Update GUI.
74    top.update_idletasks()
75    top.update()
76    # Grab Values from GUI.
77    t_run =spdW.get()
78    swingX = xW.get()
79    swingZ = zW.get()
80    start_h = stW.get()
81    inp = imuP.get()
82    ini = imuI.get()
83    ind = imuD.get()
84    # Calculate foot positions
85    # Right front foot.
86    rf_x = (swingX* (math.sin(math.radians(tik*360/t_run)-math.
    radians(90))))
87    if tik <= t_run/2:
88      rf_z = start_h + (swingZ * (math.sin((math.radians(tik*360/
    t_run))))) +imu_tf[0]
89    else:
90      rf_z = start_h +imu_tf[0]
```

```
91      # Right back foot.
92      rb_x = (swingX* (math.sin(math.radians(tik2*360/t_run)-math.
        radians(90))))
93      if tik2 <= t_run/2:
94        rb_z = start_h + (swingZ * (math.sin((math.radians(tik2*360/
        t_run))))) +imu_tf[1]
95      else:
96        rb_z = start_h  +imu_tf[1]
97      # Left front foot.
98      lf_x = (swingX* (math.sin(math.radians(tik*360/t_run)-math.
        radians(90))))
99      if tik <= t_run/2:
100       lf_z = start_h + (swingZ * (math.sin((math.radians(tik*360/
        t_run))))) +imu_tf[2]
101     else:
102       lf_xz = start_h +imu_tf[2]
103     # Left back foot.
104     lb_x = (swingX* (math.sin(math.radians(tik2*360/t_run)-math.
        radians(90))))
105     if tik2 <= t_run/2:
106       lb_z = start_h + (swingZ * (math.sin((math.radians(tik2*360/
        t_run))))) +imu_tf[3]
107     else:
108       lb_z = start_h  +imu_tf[3]
109
110
111     # Create and Send ROS message of foot positions.
112     p = Polygon()
113     p.points = [Point32(x=rf_x,y=0,z=rf_z),
114           Point32(x=rb_x,y=0,z=rb_z),
115           Point32(x=lf_x,y=0,z=lf_z),
116           Point32(x=lb_x,y=0,z=lb_z),]
117     pub.publish(p)
118
119
120
121
122
123     # Apply IMU gains from GUI values.
```

```
124     pidx.update(imu_off.x)
125     pidy.update(imu_off.y)
126 
127     imu_tf= [ (pidx.output   + pidy.output),
128               (pidx.output   - pidy.output),
129               (-pidx.output  - pidy.output)]
130               (-pidx.output  + pidy.output),
131 
132 
133     # Loop counter tik and tik2, tik2 is 180 degrees out of phase.
134     if tik < t_run:
135       tik += 1
136     else:
137       tik = 0
138     tik2 = tik + t_run/2
139 
140     if tik2 < t_run:
141       tik2 += 1
142     else:
143       tik2 -= t_run
144 
145     rate.sleep()
146 
147 if __name__ == '__main__':
148   try:
149     talker()
150   except rospy.ROSInterruptException:
151     pass
```

10.10 Launch file

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <launch>
3   <!--Start Robot Software -->
4   <!--Starting arguments and parameters.-->
5   <arg name="model" default="/home/jerome/catkin_ws/src/
     rosberry_pup/urdf/ROSberryPup2.urdf" />
6   <arg name="gui" default="false" />
7   <arg name="joint" default="[joint]" />
8   <arg name="rvizconfig" default="/home/jerome/catkin_ws/src/
```

```
    rosberry_pup/urdf.rviz"/>
  <param name="robot_description" command="$(find xacro)/xacro.py
    $(arg model)" />
  <param name="use_gui" value="$(arg gui)"/>
  <param name="rate" value="50" />

  <!--Start nodes.-->
  <node name="joint_state_publisher" pkg="joint_state_publisher"
    type="joint_state_publisher" >
     <rosparam param="source_list">["joint"]</rosparam>
  </node>
  <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="state_publisher" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg
    rvizconfig)" required="false" />
  <node name="ikpup" pkg="rosberry_pup" type="ikpup.py" />
  <node name="Planner" pkg="rosberry_pup" type="Planner.py" />
</launch>
```

11 Project Planning

11.1 Parts List

Part Name	Cost
LDX-218 lobot, robotic servos x 12	na
Adafuit Absolute orientation IMU (BN0055)	5
Raspberry Pi 3B	na
16 channel servo controller (PCA9685)	2
Ubuntu laptop	na
Shock absorber (VGEY1) x4	10
Hall effects sensor x4	5
Neodynium magnets x4	5
M3 screws and spacers	5
5mm acrylic sheets x3	8
Adafuit 16-bit ADC (ADS1115)	15
ABS/PLA 3D printing filamnt	15
Total cost	70

11.2 Gantt Chart

Table 5: Gnatt Chart

	weeks																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Reseach quadrupedal design	x 0	x 0															
Research quadrupedal motion	x 0	x 0															
Reseach quadrupedal motion	x 0	x 0															
Robot Specification			x 0														
CAD design				x 0	x 0												
Construction of robot						x 0	x 0										
Configure ROS on master and slave							 0	x									
Implement invese kinematics								 0	x 0	x							
Configure Sensors										 0	x 0						
Create gait functions												x 0	x 0				
Implement control algorithm														x 0	x 0		
Tune control algorithm																x 0	x 0

11.3 Work Breakdown Structure

Table 6: Work breakdown structure

complete: **
partialy complete: *