Design and Development of a Humanoid Robot

Project Report

Submitted to The APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

B.Tech.

in

Electrical & Electronics Engineering

Submitted by

Albin Biju (TVE18EE015)

Devika S. Nair (TVE18EE050)

Rithik S. (TVE18EE100)

Guided By

Dr. Jisha V. R.

DEPARTMENT OF ELECTRICAL ENGINEERING

COLLEGE OF ENGINEERING TRIVANDRUM

KERALA

June 2022

Declaration

We undersigned hereby declare that the seminar report **Design and Development of a Humanoid Robot**, submitted for partial fulfillment of the requirements for the award of degree of Bachelor of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by us under supervision of Dr. Jisha V. R. This submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Place : Thiruvananthapuram

Date : 30/06/2022

Albin Biju

Devika S. Nair

Rithik S.

DEPARTMENT OF ELECTRICAL ENGINEERING

COLLEGE OF ENGINEERING TRIVANDRUM

THIRUVANANTHAPURAM - 16,

Certificate

This is to certify that the report entitled ***Design and Development of a Humanoid Robot*** *submitted by* ***Albin Biju, Devika S. Nair and Rithik S.****, to the* ***APJ Abdul Kalam Technological University*** *in partial fulfillment of the requirements for the award of the Degree of* ***B.Tech in Electrical & Electronics Engineering*** *is a bonafide record of the project work carried out by them under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.*

Dr. Jisha V. R.
Professor and Guide
U.G. Coordinator
Dept. of Electrical Engg.
College of Engineering
Trivandrum

Dr. Sreeja S.
Assistant Professor
Dept. of Electrical Engg.
College of Engineering
Trivandrum

Dr. Savier J. S.
Professor and Head of the Dept. of Electrical Engg.
College of Engineering Trivandrum

Acknowledgement

We have great pleasure in expressing our gratitude to **Dr. Savier J. S.**, Professor, Head of the Department, Department of Electrical Engineering, College of Engineering, Trivandrum for his valuable guidance and suggestions to make this work a great success.

We would like to express our sincere thanks to the thesis guide **Dr. Jisha V. R.** , Professor, Department of Electrical Engineering, College of Engineering, Trivandrum for all the necessary help extended to us in the fulfilment of this work.

We also express our gratitude to **Dr. Sreeja S.**, Assistant Professor, Department of Electrical Engineering, College of Engineering, Trivandrum, for all the guidance and encouragement in the fulfilment of this project. We also acknowledge our gratitude to other members of faculty in the Department of Electrical Engineering, our family and friends for their whole hearted cooperation and encouragement.

ALBIN BIJU

DEVIKA S. NAIR

RITHIK S.

Abstract

Biped robots have better mobility than conventional wheeled robots, but they tend to tip over easily. To be able to walk stably in various environments, such as on rough terrain, up and down slopes, or in regions containing obstacles, it is necessary for the robot to have a strong and composite mechanical structure that can adapt to the ground conditions with a foot motion, and maintain its stability with a torso motion. The main objective of this project is to design a stable mechanical structure of a 50 cm tall lower limb model of a Bipedal humanoid robot from scratch. The parts and 3D model of the structure was designed in Solid Works and was implemented in hardware using Aluminium 6063 Alloy and appropriate actuators. Finally, the hardware model was stabilised using inverse kinematics technique. The prototype developed was also able to implement a stable walking gait when tested with a valid walking trajectory.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

A robot is a machine that is able to perform tasks autonomously. These tasks can be very diverse and are generally intended to take over human tasks. Robots can be divided into three types: *industrial robots*, *service robots* and *humanoid robots*. Industrial and service robots are intended to perform one specific task like in an industrial environment.

A humanoid robot can be the design of a specific body part to even an imitation of the total human beings (bio-) mechanics [1]. One of the most important application of humanoid robotics is in the development of prosthetic devices. Inorder to develop devices that helps replace, correct, or support a body part or function of a body part, we must be well versed in the human body motion and dynamics. Study of humanoid robots helps in improving this idea and helps in the design and implementation of highly efficient prosthetic systems.

Another major application of humanoid robotics is the development of medical robots for elderly care. Using medical robots for elderly care will vastly reduce the current astronomical cost of elderly care. Additionally, it will pick up the slack in terms of the number of caregivers available as the ratio of elderly to

nonelderly people shifts. Medical robots are a growing industry. According to the International Federation of Robotics World Robotics 2018 Service Robots report, medical robot sales increased 73 % in 2017 over 2016, accounting for 2.7% of all professional service robot sales.

Bipedal robots are able to move in areas that are normally inaccessible to wheeled robots and areas filled with obstacles that make wheeled locomotion impossible. An uneven floor has to be made flat, a narrow passage should be removed and a lift must be available for a robot on wheels. Also it may be easier for people to interact with walking robots with human morphology rather than robots with non-human shape. Robots should be able to do tasks alongside humans being our partners enjoying communications. These days humanoids also finds scope in amusement and entertainment.

1.2 Motivation

Humanoid robots are one of the captivating autonomous systems in human society as they can work well in indoor environment designed for humans. There are different classes of robots namely : the wheeled robots, spherical robots, Biped robots etc. Humanoid robots fall under the category of biped robots. Unlike the other categories of robots, the biped robots are designed to imitate human behavior to move and step in any arbitrary terrain. The structural modelling of a humanoid robot is done by analysing the anatomy of human body so that each joint movement is carefully replicated. The main reason why humanoids are still not part of our society is because of their mobility in a human environment. Our environment is based on moving around on two legs, often called biped locomotion, like walking. Although walking seams effortlessness for humans, it is one of the most complex movements to imitate. The development of humanoid walking is still at its early stages because we simply do not know exactly how

humans walk and how every muscle is used.

One of the major factors concerning the stability in biped locomotion is the rigidity and robustness in the structural design. The development of (anthropomorphic) [1] humanoid robots is still in its infancy. Compared to human beings, all existing humanoid robots exhibits the following features:

- too simplified kinematic structure
- too heavy
- move around for a short amount of time due to poor energy sources (batteries)
- use simple control schemes for walking and posture stability
- use less sensors to examine their surroundings
- have less computational power

1.3 Objectives

- To design and develop a 50cm tall lower limb structure of a bipedal humanoid robot.
- Design of structure of limbs which mimics human motions.
- Realisation of kinematic model for a robot with natural walking gait.
- Finding the optimum range of torques, speeds and angles for the joints.
- Proper selection of actuators and materials for a stable structural design.
- Stabilising the model.

1.4 Outline of the Report

This project report is organised as follows. Chapter 1 discusses the general trends in robotics, need for the development of humanoid robots, its current social and industrial status and its applications. In Chapter 2, A literature review is conducted based on the basic structural modeling of humanoid robots. Chapter 3 describes the basic idea of the project and shares a brief idea about the implementation strategies adopted. The methodology adopted for proceeding and various theories related to this work is detailed in Chapter 4. Chapter 5 elaborates the hardware design strategies adopted. Software design and control algorithm generation is explained in chapter 6. The results and inferences from the work is presented in chapter 7. Finally, the report is concluded with the conclusions and future scope in chapter 8.

Chapter 2

Literature Review

The advances in robotics and the expanding significance of humanoid robots led to a growing business interest in this field. A humanoid robot features the body shape analogous to that of human being. Generally, *Anthropometry* [1]- the science of obtaining systematic measurements of the human body is used for designing the link dimensions and overall structural features of a humanoid robot. This makes it possess the locomotion and manipulation capabilities similar to human being. Humanoid robot TUlip [1] is a humanoid robot developed by the combined efforts of three universities of the Netherlands (*University of Twente UT, Delft University of Technology TUD and Eindhoven University of Technology TU/e*). It consisted of 6 DOFs per leg: 3 at the hip joint, 1 at the knee joint and 2 at the ankle joint. This makes a total of 12 DOFs for the lower body. Also, the link dimensions of The TUlip was estimated using anthropometric calculations.

It is expected that a humanoid robot may work or assist people in the human-centered environment without a need to adapt itself or to modify the environment. To achieve fast tasks, a humanoid robot should possess the property of high stiffness, light weight and inertia, high power output, and more rigorous real-time information sensing and processing capability. Recently, many researches have been focused on the development of humanoid biped robot. Honda R&D's

humanoid robots [2], ASIMO [3], Partner, QRIO, H6 & H7 [4], HRP [5] and JOHNNIE [6] are some of the well known humanoids. The humanoid robot WABIAN-2, which was developed at Waseda University, has 7 DOF for each leg, 2 DOF for the waist, 2 DOF for the trunk, and 7 DOF for each arm [7]. This robot has more redundancy in the upper body, arms, and legs than a conventional biped humanoid robot, enabling it to move in various ways such as walking, hand shaking, and bowing.

Since the humanoid is very complicated, expensive and unstable, it is difficult to construct the mechanical body itself, to integrate its hardware system, and to realize a real-time motion and stability control based on the sensory feedback similar to human behavior.There are different control algorithms proposed to different type of robots. Each control algorithm has its own pros and cons. The selection of the control algorithm depends on the type of robot to be developed and the number of degrees of freedom(DOF). One of the most important control strategy is based on the concept of Zero Moment Point (ZMP). When the robot is falling down, the sole of the supporting foot should not contact with the ground any more. The ZMP (Zero Moment Point) proposed by Vukobratovi´c et al. [8] is a criterion to judge if the contact between the sole and the ground can be kept without solving the corresponding equations of motions. The contact is kept if the ZMP is an internal point on the sole. When the robot does not move, the contact is kept when the projection of the center of the mass of the robot onto the ground is an internal point of the sole.

Chapter 3

System Description

Biped Robots are a subclass of legged robots that can be defined as an open kinematic chain which consist of two subchains termed as legs, each connected at a common point termed hip. In this project, an empirical structure of the human form is chosen as the fundamental mechanical design.

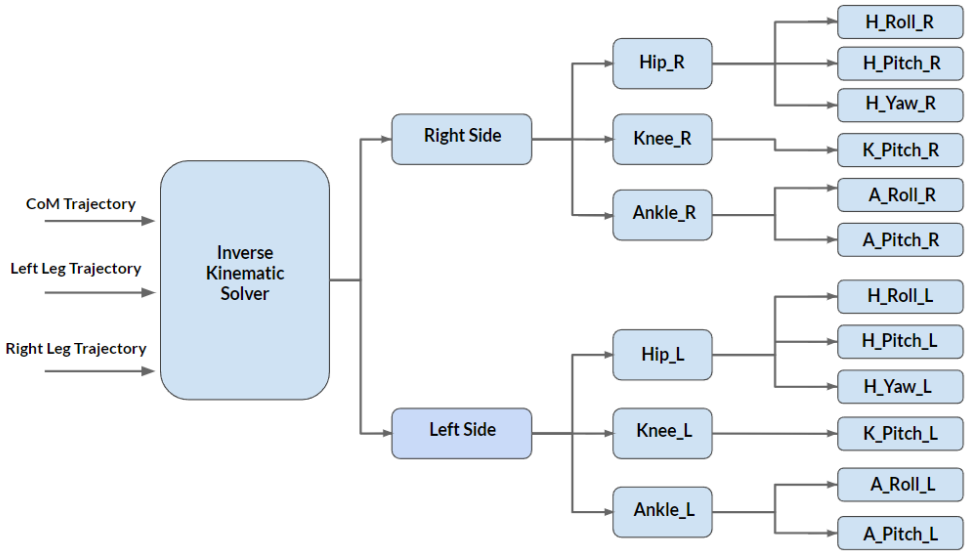

Figure 3.1: Basic System Block Diagram

A kinematic configuration of 6+6 DoF is chosen so that the basic human movements can be replicated. It is also desired that the model must be able to implement a stable walking when provided with a valid walking trajectory. Inorder to achieve this, an inverse kinematic controller is also designed. The

block diagram of the basic system model is illustrated in figure 3.1.

When supplied with a valid walking trajectory, the inverse kinematic solver generates suitable joint angles inorder to implement the walking gait and is passed on to the two sub chains - the right and left legs.

The distribution of DoFs for each joint in each leg is illustrated in the system block diagram (figure 3.1).

3.1 Robot Structure

Inorder to mimic the human walking motion and other activites, the proposed model is designed with three joints (Hip, Knee & Ankle) for each leg, connected by links of appropriate dimension. A human leg provides 7 degrees of freedom. But, the human movements like flexsion, extension, abduction, internal rotation and external rotation can be realised by choosing minimum 6 Degrees of Freedom for each leg. Hence in total, the proposed model is designed with 12 DoF. The detailed distribution of the DoF's among the 3 joints of each leg is illustrated in figure 3.2.

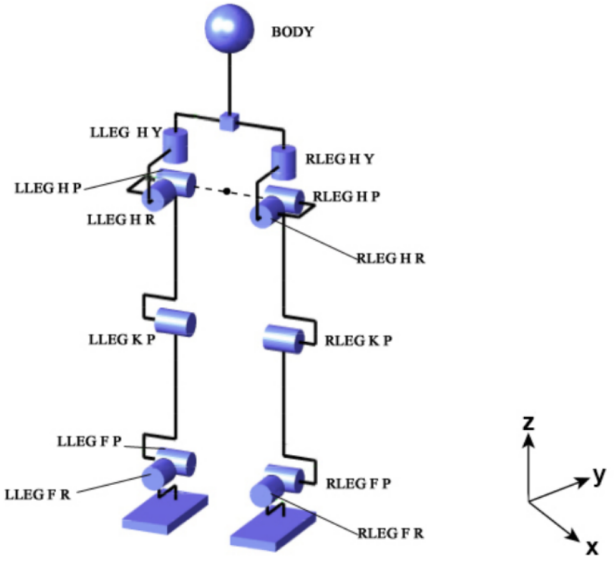

Figure 3.2: Joint Diagram

Human hip consists of a ball and socket joint. To mimic the joint, 3 DOF

joint is designed to provide the roll, pitch and yaw motions. Knee joint is limited to 1DOF. Knee motion is limited to only one direction from the mean position. The knee joint is similar to the hinge joint and the knee joint rotation is also limited. Pitch and Roll motions at feet is required for human gait. Human has pitch, roll and yaw motions at feet. Due to the difficulty in implementation and cost constraints of 3DOF at feet, yaw motion is dropped and only 2DOF is implemented.

This 12 DoF structural design implements the human motion consistently when provided with appropriate joint angle limits and a suitable control strategy.

Chapter 4

Methodology

This chapter encompasses the theoretical analysis of the methods and concepts adopted in the project. It includes the basics of human motion and the ways of imitating it along with the theories behind the structural modelling of the robot and its motion analysis.

4.1 Mechanics of Human Movement

The human body has exceptional abilities to generate flexible, powerful, quick, and agile movement, accomplishing a variety of tasks such as flexible, agile locomotion, dexterous grasping, and manipulation. The ability to reproduce at least a subset of human motion capabilities has long been the goal of robotics. The correspondence problem appears to be more readily solved, as there is clear correspondence between human and humanoid body parts. However, due to differences in size, range of motion, and range of achievable joint torques, correspondence for the full range of human movement may still be a significant issue. Unlike fixed base robots, a major additional task during imitation is to maintain postural balance, a task which can interfere with the goals of accurate motion imitation. A final challenge particularly relevant to humanoid motion imitation is, how to evaluate the imitation and specifically to evaluate to what

extent the generated robot motion is humanlike.

A biped can be positioned in the three dimensional space with a base- frame-origin and three mutually perpendicular planes (Figure 4.1). The plane parallel to the yz-plane is called the Frontal plane. The plane parallel to the xz-plane is called the Sagittal plane. The plane parallel to the Sagittal plane and containing the Center of Mass (CoM) is called the Median plane.

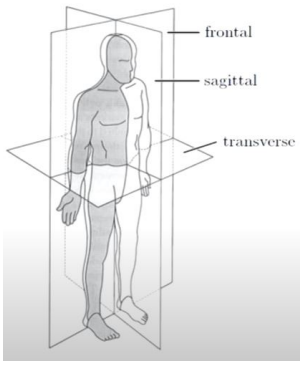

Figure 4.1: Mutually perpendicular Planes

Figure 4.2 shows the maximum joint angle limits for different motions in the human body. These joints are provided for a larger set of applications and are not specifically for walking.

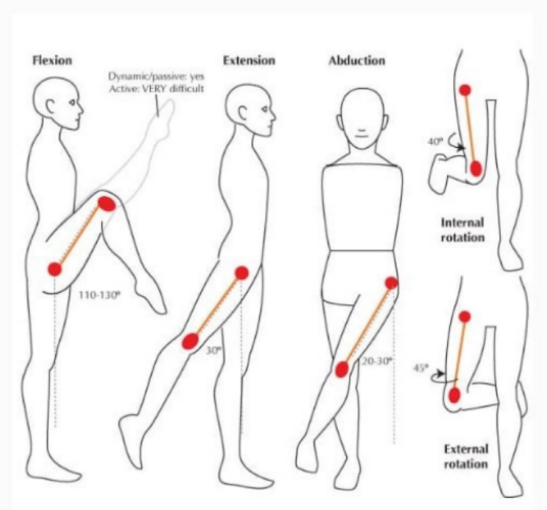

Figure 4.2: Limits of joint angles for various human motion

The basic human joint motions like flexion, extension, abduction, internal and external rotation as illustrated in figure 4.2 can be implemented in robotic

joints in 3 dimensional plane. Robotic joints typically used in humanoid robots are either rotary (revolute) or linear (prismatic). A revolute joint is like a hinge and allows relative rotation between two links. A prismatic joint allows a linear relative motion between two links. The Symbolic representation of robot joints are illustrated in figure 4.3.

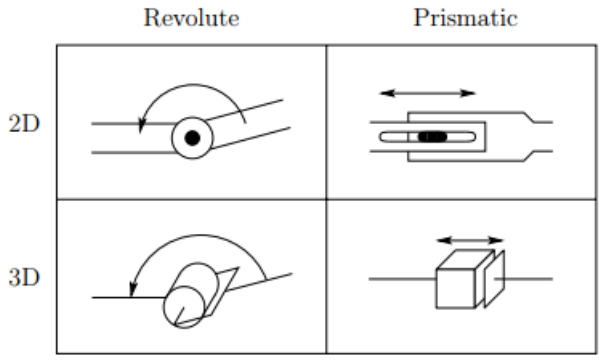

Figure 4.3: Symbolic Representation of Robot Joints

4.2 Human Motion Imitation

The complete processing pipeline for human motion imitation can be divided into four stages:

1. Human motion must be observed and recorded. For highest accuracy, motion capture is typically used where, small lightweight markers are placed on human body landmarks and their 3D position recorded by a set of cameras. The resulting dataset consists of the Cartesian trajectory of each body landmark over the duration of the motion.

2. The motion data is preprocessed to extract the relevant motion segments, remove noise and interpolate missing marker data, and convert the data into a format suitable for imitation. A typical conversion is inverse kinematics, where Cartesian data is converted to joint angles by using the kinematic model of the robot or the human demonstrator.

3. A model of the movement (or movements) is created.

4. Finally, using the developed model and additional control elements, the motion is reproduced on a humanoid platform and the resulting motion evaluated.

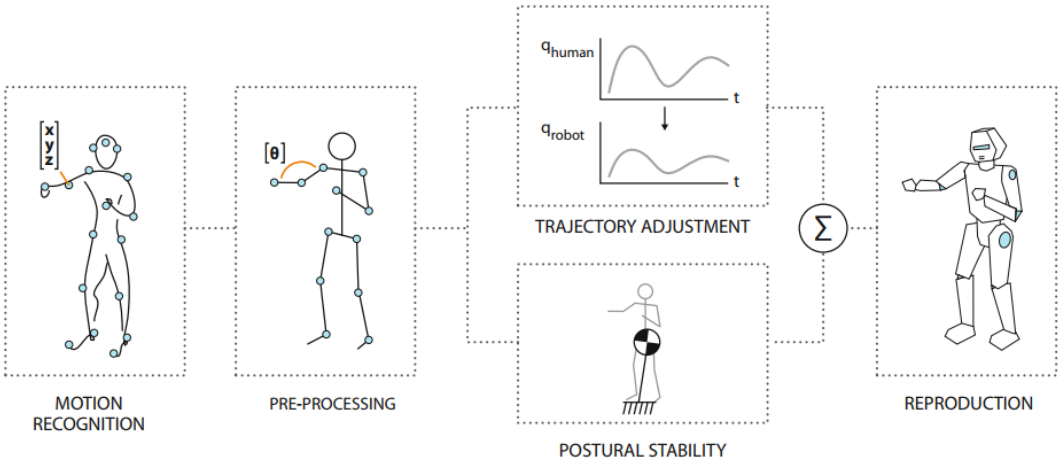

Figure 4.4: General processing pipeline for Human Motion Imitation

Figure 4.4 represents the direct reproduction with separate postural control approach [9] where, joint angle time series obtained from human motion capture is used directly as the reference trajectory, and a control strategy is formulated so that the robot follows the human joint angle trajectory as closely as possible while maintaining postural balance.

4.3 Basics of Humanoid Gait Cycle

Gait Analysis For studying the human gait cycle one needs to be familiar with some specific terms, which are defined below. Figure 4.5 shows the human gait cycle.

- **Gait:** It is the pattern of limb movements made during locomotion.
- **Periodic Gait:** A gait that is performed by repeating the steps in an identical way, it is a periodic gait

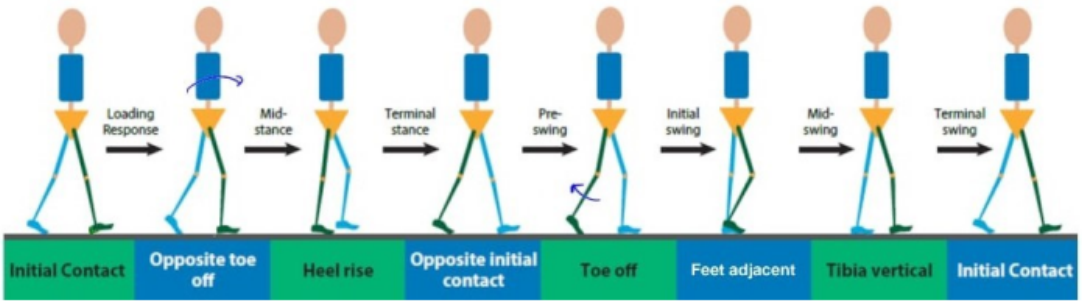

Figure 4.5: Human Gait Cycle

- **Walk:** Walk is the movement by putting forward each foot in turn, not having both feet off the ground at the same time.

- **Double Support:** This term is used for situations where the biped has two isolated contact surfaces with the floor. This situation occurs when the biped is supported by both feet, but it is not necessarily that both feet are fully supported with the floor.

- **Single Support:** This term is used for situations where the biped has only one contact surface with the floor. This situation occurs when the biped is supported with only one foot.

- **Support Polygon:** The Support Polygon is formed by the convex hull about the floor support points. This term is widely accepted for any support area.

- **Swing leg/foot:** The leg that is performing a step (moving forward through the air) is denoted with the term swing leg. The foot that is attached to this leg is called the swing foot.

- **Stance leg/foot:** While the swing leg is moving through the air, the stance leg is fully supported with the floor by the stance foot and supports all the weight of the biped.

- **Gait Phases:** When the biped is in periodic gait, the gait can be divided

into four phases:

1. **Double Support Phase (DSP):** This is the phase where both feet are fully supported with the floor.
2. **Pre-Swing Phase:** In this phase the heel of the rear foot is lifting from the floor but the biped is still in double support due to the fact that the toes of this foot are still on the floor.
3. **Single Support Phase (SSP):** The phase where only one foot is fully supported with the floor and the other foot swings forward.
4. **Post-Swing Phase:** In this phase the toe of the front foot is declining towards the floor. The biped is in double support because the heel of this foot is contacting the floor.

4.4 Structure and Rigid Body Tree

The lower body structure designed is 50 cm tall with the body parts of dimensions according to Anthropometry. The lengths were determined using the above method whereas the other details were determined based on the strength and rigidity of the brackets that were designed to connect between two joints. The dimensions between two joints were implemented in the robot configuration and the motions required at each joint were decided and thus the Kinematic configuration was made. The parts were modelled in Solid Works and structural analysis was performed.

4.4.1 Link Dimension Design

The humanoid robot model is to be developed as a scaled down version of the human body. Hence, anthropometric dimensions can be used to design the links and joints. The human skeleton can be taken as an example for the link

dimensions of a humanoid robot [1].

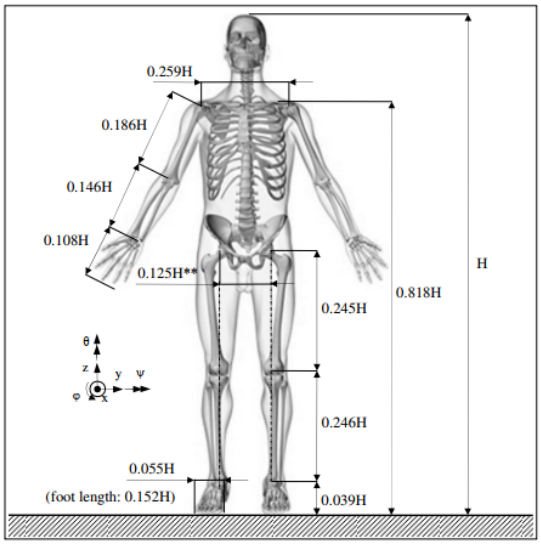

Figure 4.6: Anthropometric Design

In figure 4.6, the key dimensions of the human skeleton are illustrated.

4.5 Kinematics

The theory to analyze the relationship between the position and attitude of a link and the joint angles of a mechanism is called Kinematics. It is the basis on which robots are analysed. Position and orientation of each link is to be known in order to do forward kinematics. We first define an origin for the world coordinate. This coordinate system is named as W. The X-axis of the world coordinate faces forward direction. Z-axis is along the height of the robot. Y-axis is directed along the left. Absolute position, absolute velocity, and absolute rotation of each link is defined with respect to the world coordinate frame. A local coordinate system is defined for each link of the robot. Chain rule of homogeneous transformation is used to connect different local coordinates carefully to find the position and orientation of a point in world coordinates. Chain rule reduces the complexity of kinematic calculations.

4.5.1 Link Tree Structure

A humanoid robot is a mechanism consisting of many links connected by joints. Inorder to make processing easy, we first conceptually separate the entire structure into smaller units. The robot structure is divided in such a way that each joint gets included with the link that follows it. For example, instead of associating knee joint with the upper leg, we associated it with the lower leg. It was to make sure that every link will get only one joint. It helps to use the same code for every joint. It is to be noted that this separation is only a conceptual one. Our humanoid robot has 12 DoF, 6 on each leg and each link has a name. Now in order to connect each link to get a whole humanoid robot, we use a connection rule as shown in the Figure 4.7. The connection rule we used is similar to a family tree. Left arrow of each link indicates its child link and right arrow indicates the sister link. This family structure can make use of recursive programming to access all links.

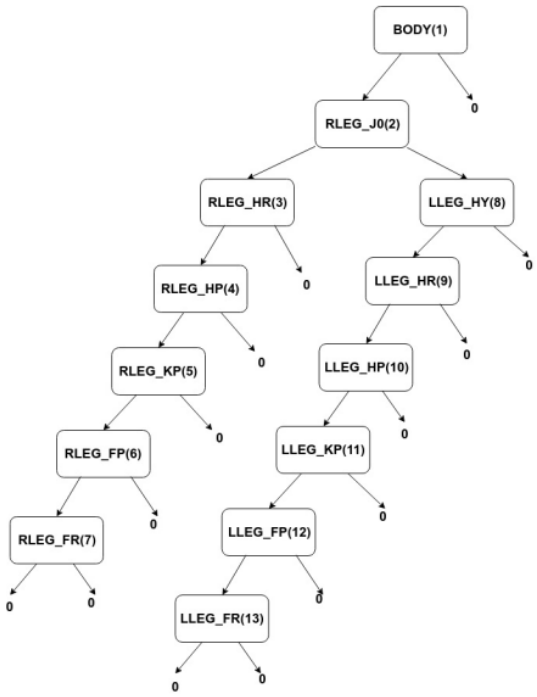

Figure 4.7: Link Family Tree

Once we have the link family tree established, the next step is to define the

local coordinate system for each link and assign rotation matrices. Initially, we set these coordinate systems to match that of world coordinates. So we set 13 rotation matrices as, R1 = R2 = R3 = R4 = R5 = R6 = R7 = R8 = R9 = R10 = R11 = R12 = R13 = E, where E is the unit matrix. We defined joint axes along the axis of rotation of each joint.

4.5.2 Forward Kinematics

Forward Kinematics is a calculation to obtain the position and attitude of a certain link from a given robot structure and its joint angles. This is required to calculate the Center of Mass of the whole robot, when we just want to display the current state of the robot, or to detect collisions of the robot with the environment. Thus, forward kinematics forms the basis of robotics simulation. Forward kinematics can be calculated by using the chain rule of homogeneous transforms. First, the homogeneous transform of a single link is calculated. The

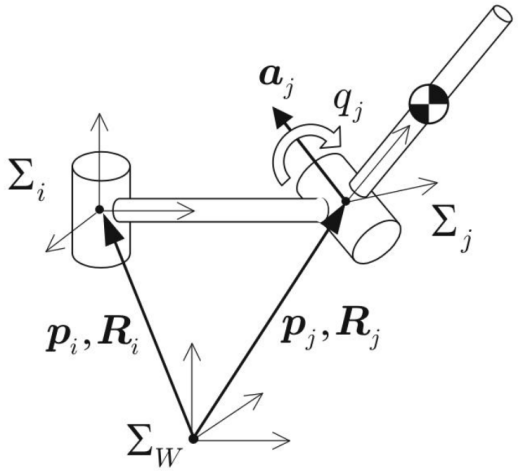

Figure 4.8: Relative positions of 2 links

homogeneous transform of a link relative to its parent link is given by:

$$T_j^i = \begin{bmatrix} & e^{\hat{a}_j q_j} & & b_j \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where, a_j is the joint axis vector seen from the parent coordinates and b_j is the origin of $\sum_j$. q_j is the joint angle. Now considering two links as shown in figure 4.8, the homogeneous transform of $\sum_i$ becomes,

$$T_i = \begin{bmatrix} & R_i & & p_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where, p_i and R_i are the position and attitude of the i^{th} link. From the chain rule of homogeneous transform $\sum_j$ is

$$T_j = T_i T_j^i$$

The absolute position p_j and attitude R_j of $\sum_j$ can be calculated using:

$$p_j = p_i + R_i b_j$$

$$R_j = R_i e^{\hat{a}_j q_j}$$

Similarly, the velocity of a link connected to its parent link can also be computed, given the joint speed $\dot{q}_j$, its linear velocity is v_j and angular velocity w_j.

$$v_j = v_i + w_i * R_i b_j \quad ; \quad w_j = w_i + R_i a_j \dot{q}_j$$

4.5.3 Inverse Kinematics

In order to calculate joint angles when we have the position and attitude of the body and the foot to realize. So in this case Inverse Kinematics is the right choice. For instance, suppose our robot is in the front of stairs and we want to place one of the foot on the first step whose height and depth are already known. We certainly need to determine the amount of joint rotation for the hip, knee and the ankle. Inverse Kinematics is necessary for such a case. Analytically

solving this can be shown by the block diagram shown in figure 4.9. There exist

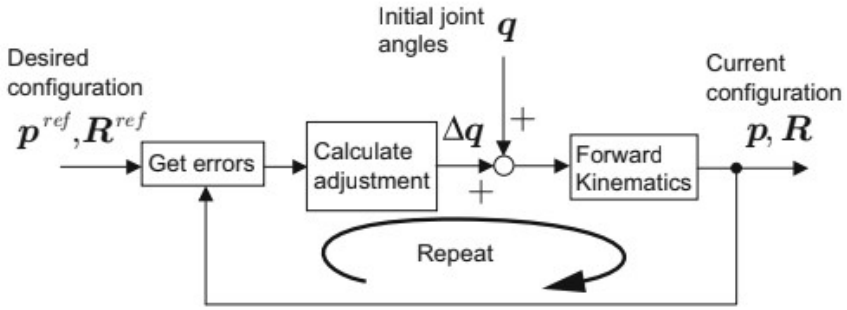

Figure 4.9: Basic Concept Behind Numerical Approach to Inverse Kinematics: Use forward kinematics to adjust the joint angles to narrow the difference.

both an analytical method and a numerical method of solving Inverse Kinematics but in general implementation of analytical solution requires a large amount of heavy calculation, so it is more common to use the numerical solution. Compared to solving the inverse kinematics analytically, forward kinematics calculation is simple. So it isn't all that far fetched to think of a trial and error method of solving the inverse kinematics by using forward kinematics, as shown in fig 4.9. A sample algorithm could be,

- Prepare the position and attitude $\boldsymbol{p^{ref}}, \boldsymbol{R^{ref}}$ of the base link
- Prepare the position and attitude $\boldsymbol{p^{ref}}, \boldsymbol{R^{ref}}$ of the target link
- . Define vector $\boldsymbol{q}$ which holds the joint angles from the base link to the target link
- Use forward kinematics to calculate the position and attitude $(\boldsymbol{p}, \boldsymbol{R})$ of the target link
- Calculate the difference in position and attitude $(\boldsymbol{\Delta p}, \boldsymbol{\Delta R}) = (\boldsymbol{p^{ref}} - \boldsymbol{p}, \boldsymbol{R^T R^{ref}})$
- If $(\boldsymbol{\Delta p}, \boldsymbol{\Delta R})$ are small enough stop the calculation
- If $(\boldsymbol{\Delta p}, \boldsymbol{\Delta R})$ are not small enough calculate q which would reduce the error
- Update joint angles by $\boldsymbol{q} := \boldsymbol{q} + \boldsymbol{\Delta q}$ and return

4.6 Center of Mass

The main challenge in the walking control of a biped humanoid robot is the stabilization. Since controlling the robot's Center of Mass (CoM) provides significant aid in maintaining static balance given the complexity of the problem. In order to achieve reliable estimation of humanoid robot Center of Mass (CoM) motion, we need a model that explains how the CoM moves under external forces. The Linear Inverted Pendulum Model (LIPM) model is a natural choice given its simplicity. We introduce an additional offset term into the LIPM dynamics. This offset can be interpreted as a modelling error on the CoM position, or an external force exerted on the CoM of the robot, or a combination of both.

4.7 Zero Moment Point

In figure 4.10 an example of force distribution across the foot is given. As the load has the same sign all over the surface, it can be reduced to the resultant force R, the point of attack of which will be in the boundaries of the foot. Let the point on the surface of the foot, where the resultant R passed, be denoted as the zero-moment point, or ZMP in short.

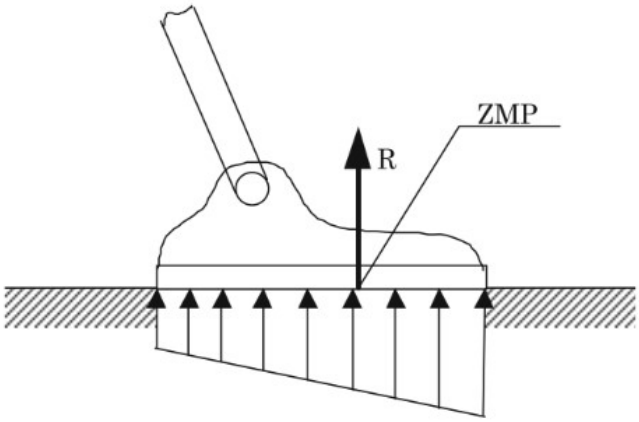

Figure 4.10: Definition of Zero-Moment Point (ZMP)

We explain the support polygon which is another important concept related to

the ZMP. As shown in figure 4.11, let us consider the region formed by enclosing all the contact points between the robot and the ground by using an elastic cord braid. We call this region as the support polygon. Mathematically the support polygon is defined as a convex hull, which is the smallest convex set including all contact points.

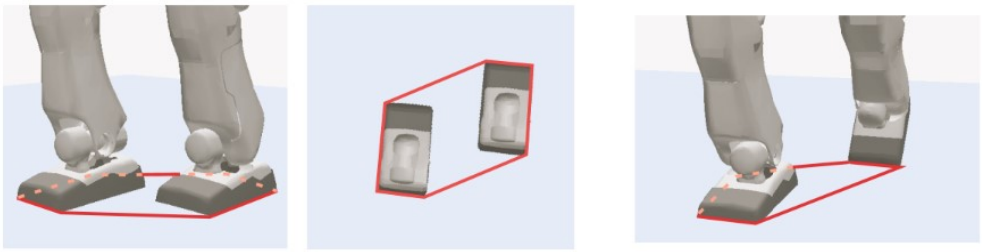

Figure 4.11: Support polygon for full contact and partial contact of feets

In order to illustrate this more concretely, consider the images in figure 4.12 illustrating the relationship among the center of mass (CoM),ZMP and the support polygon while a human stands on the ground. We call the ground projection of CoM the point where the gravity line from the CoM intersects the ground. As shown in figure 4.12, when a human stands on the ground, the ZMP coincides with the ground projection of CoM. In such a case, a human can keep balance if the ground projection of CoM is included strictly inside of the support polygon. On the other hand, when a human moves dynamically as shown in figure 4.12, the ground projection of CoM may exist outside the support polygon. However, the ZMP never exists outside the support polygon.

The approximation of a humanoid robot by an inverted pendulum is one of the most frequently used models to generate a stable walking pattern using a planned zero moment point (ZMP) trajectory.

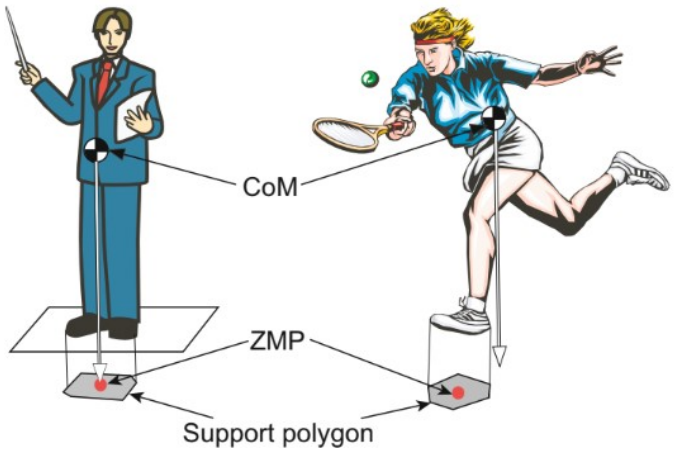

Figure 4.12: COM, ZMP and Support polygon of standing human and human in action

4.8 Static Stability

Biped robot walking based on the zero moment point (ZMP) has been widely applied and achieved remarkable achievements. Given dynamic degree of biped walking, according to the relation between projection of center of mass (CoM), ZMP, and supporting convex polygons, walking can be divided into static walking, quasi-dynamic walking, and dynamic walking. In static walking, the projection of CoM of robot on the ground never exceeds the range of supporting polygons. In dynamic walking, the projection of CoM can exceed the supporting polygon at some point, which requires online control of CoM to generate stable walking pattern. Predictive control is mainly used to control CoM motion to generate stable walking pattern and achieve tracking of expected ZMP trajectory by practical ZMP trajectory. The principle of controlling CoM motion to generate stable walking pattern is also consistent with the requirement of adjusting CoG during human walking. Predictive control is to use future target ZMP information, which is consistent with the principle that human walking uses future road information to achieve precise control of CoM motion. In order to achieve static stability for the biped the COM must lie inside the support polygon. As the height of COM reduces during stand still the more stable the system becomes.

This even helps in achieving easier lifting of the legs.

Thus, in this chapter we have discussed the human motion imitation and basic theory behind structural modelling. The detailed design of the parts and structure shall be taken up in the forthcoming chapters.

Chapter 5

Hardware Design

Design of the bipedal structure requires careful examination and analysis of various factors like link dimension design, joint angle limits, torque requirement for actuation etc. Hence, the work is divided into three phases. As the first phase, a primitive model of the proposed structure is designed and simulated in Simscape Multibody by taking into consideration all the key factors of design as mentioned in chapter 4. In the Phase - 2 of the work, the actual structure is modelled and URDF file is generated. Also, the detailed design of the different parts is finalised, simulated and fabricated for hardware impementation. In Phase - 3, the hardware implementation of the model is completed and static stability in standing pose is tuned and verified

5.1 Primitive Model

As a first step towards model design, a primitive model of the entire structure was developed. The joint structure and DoFs were followed as mentioned in the basic system model. The link dimensions were calculated based on the anthropometric data as mentioned in Section 4.4.1. In figure 4.6, the link dimensions are expressed as fractions of the total height H of the entire humanoid robot structure. But, since we are constructing only the lower limbs of the humanoid robot, and have

already fixed the height of the legs to be 50cm, we calculate the rest of the link dimensions accordingly as follows :

- Height of the lower body = 50cm = 0.5m

  ie; 50 = (0.245 + 0.246 + 0.039) x H

  Hence, H = 94.339cm = 0.94339m

From this calculation, we can infer that the entire structure of the humanoid including the torso would be 94.339cm. Now, using this we can calculate the rest of the link dimensions.

- Length of Upper link of the leg = 0.245 x 0.94339 = 0.23113m
- Length of Lower link of the leg = 0.246 x 0.94339 = 0.23207m
- Height of the Ankle from the foot = 0.039 x 0.94339 = 0.03679m
- Width of the Torso = 0.125 x 0.94339 = 0.11792m
- Length of the foot = 0.152 x 0.94339 = 0.14339m
- Width of the foot = 0.055 x 0.94339 = 0.05188m

Simulation

The primitive system model is simulated in Simscape Multibody with the calculated link dimensions as in figure 5.1 and basic strucure model as per section 3.1. The assumptions made for the simulation are as follows:

- The link dimensions are selected based on the anthropometric calculations
- As per the basic system model, 6 DoF are assumed on each leg
- The angle covered by each leg wrt the vertical at the hip joint is 45^0
- Frictional forces and contact with the groung is assumed to be absent

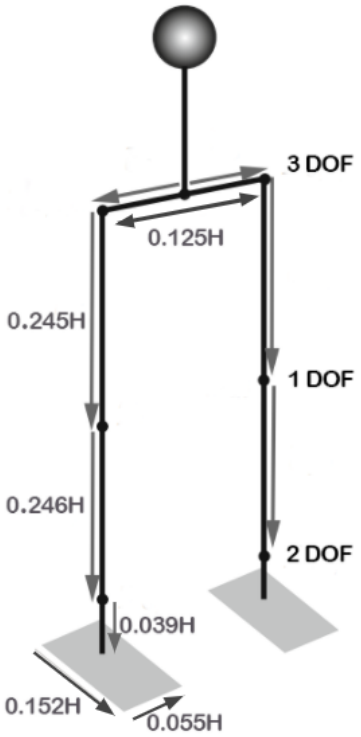

Figure 5.1: Primitive Model - Link Dimensions

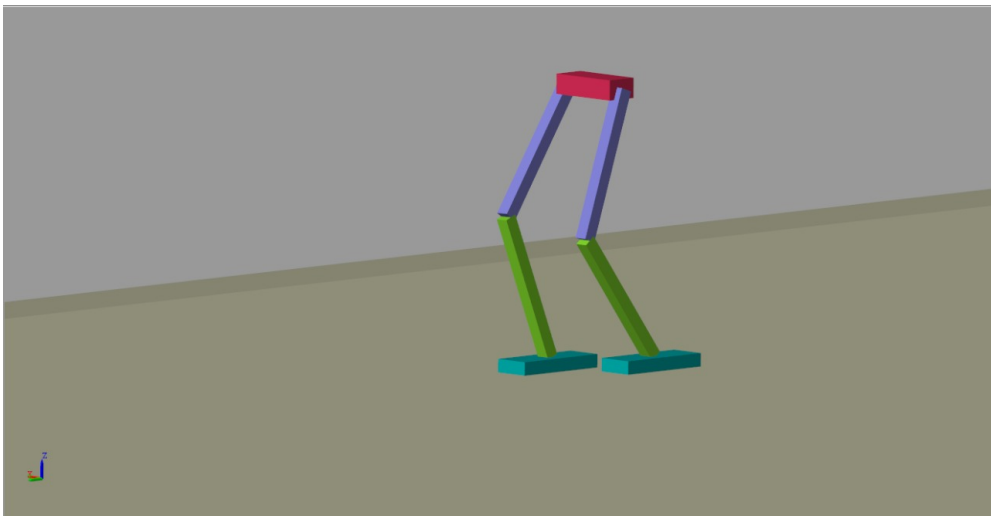

Figure 5.2: Primitive Model

The primitive model designed in Simscape Multibody is shown in figure 5.2.

The simulink block diagram of the primitive model is shown in figure 5.3. Here, the world coordinate is defined, world plane and the leg susbsystems and torso block are configured. Each block (ie; world plane, torso, leg subsystems) are assigned a local coordinate system and their position and orientation are defined

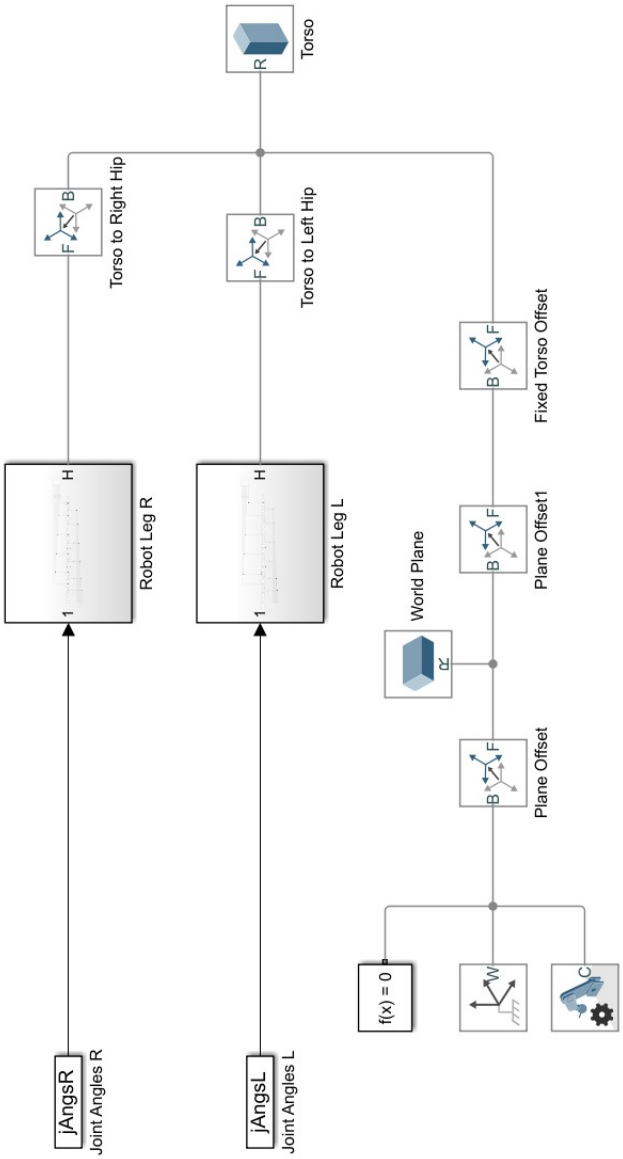

Figure 5.3: Simulink Block Diagram of the System

with respect to the world coordinate system using homogeneous transformations.

The leg subsystem simulink block diagram is shown in figure 5.5. Three blocks are used to represent the foot, lower leg and upper leg. 6 revolute joints are included as there are 6 DoF. There are 2 revolute joints between the foot and lower leg, as the ankle has 2 DoF. The upper leg and lower leg are connected using one revolute joint, as the knee has only one DoF. Beyond the upper leg

portion, we have the remaining 3 revolute joints representing the 3DoF hip joint.

The model is provided with some arbitrary angles as input and the corresponding torques at each revolute joint is plotted in figure 5.4.

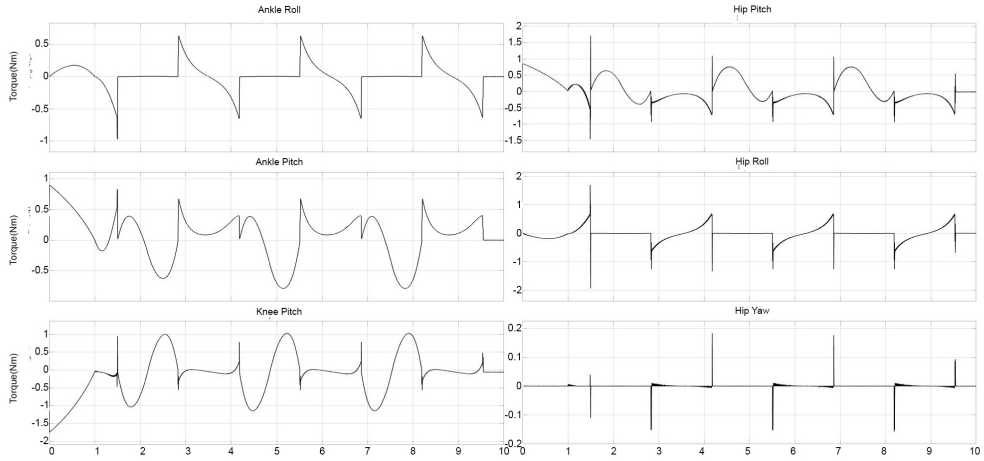

Figure 5.4: Torque requirement for the leg joints

From the graph, we can infer that the maximum torque values are 1.8 Nm at the Hip pitch and roll and 1 Nm at the Knee pitch joint. Also, the minimum torque value occurs at the hip yaw joint that is 0.18 Nm. Considering the impact that might occur while walking ie; the ground reaction force, friction etc, a thumb rule of 1.5 times the obtained torque values are calculated. Hence, the torque values fall in the range 0.27 Nm - 2.5 Nm and the actuators are selected accordingly.

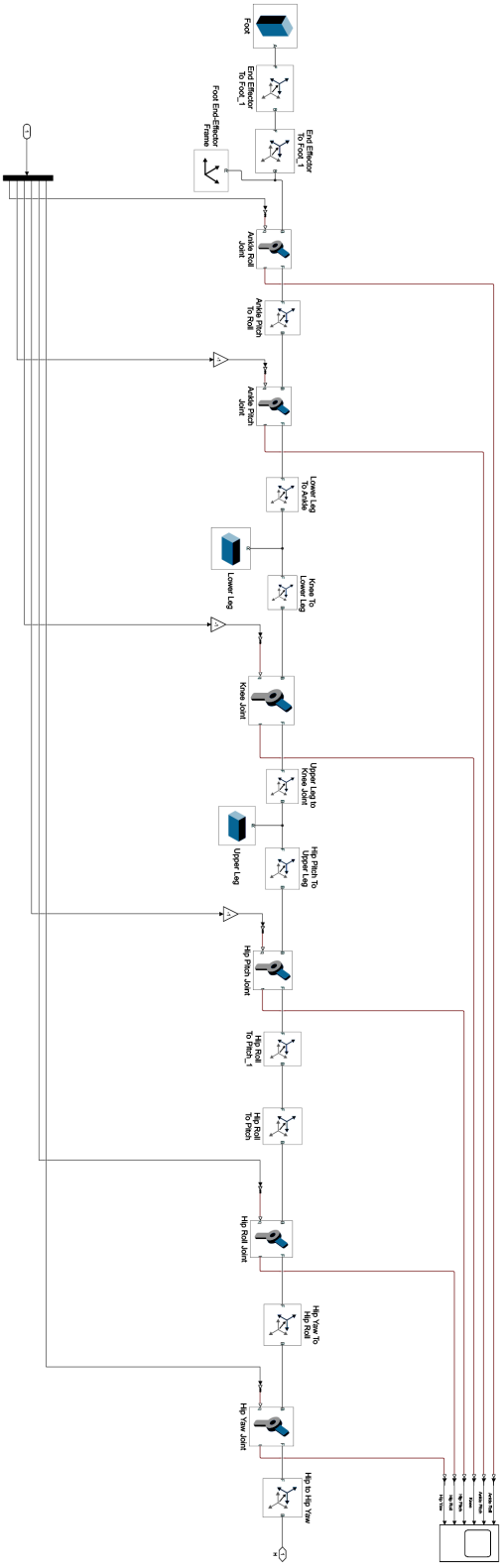

Figure 5.5: Simulink Block Diagram of Leg Subsystem

5.2 Actuators

Selection of motors is one of the most important task in designing a robot. The motors must be selected in such a way that it has good torque characteristics, good communication capabilities, light in weight, and most importantly precision in actuation.

The DYNAMIXEL is a smart actuator system developed to be the exclusive connecting joints on a robot or mechanical structure. Their design is very compact, modular and daisy chained. Some of the most important characteristics of this high performance actuator is that it has built in :

- fully integrated DC motor
- reduction gearhead
- internal controller
- motor driver circuitry
- daisy chainable network connection

The actuator status can be read and monitored through a data packet stream. There are a variety of series of dynamixel motors namely Dynamixel AX-12A, Dynamixel RX-28 etc. Based on the torque output of the primitive model, appropriate model of dynamixel motors are selected for each DoF and its distribution is shown in table 5.1. The control algorithm used to maintain shaft position on the actuators can be adjusted individually for each servo, allowing you to control the speed and strength of the motor's response. All of the sensor management and position control is handled by the servo's built-in microcontroller.

	Joints	Motor Selected	Stall Torque (Nm)	Voltage (V)
Right Leg	Hip-Roll	MX-28T	2.5	9-12
	Hip-Yaw	AX-12A	1.5	9-12
	Hip-Pitch	MX-28T	2.5	9-12
	Knee-Pitch	MX-28T	2.5	9-12
	Ankle-Roll	AX-18A	1.8	9-12
	Ankle-Pitch	AX-18A	1.8	9-12
Left Leg	Hip-Roll	MX-28T	2.5	9-12
	Hip-Yaw	AX-12A	1.5	9-12
	Hip-Pitch	MX-28T	2.5	9-12
	Knee-Pitch	MX-28T	2.5	9-12
	Ankle-Roll	AX-18A	1.8	9-12
	Ankle-Pitch	AX-18A	1.8	9-12

Table 5.1: Motor Selection

5.2.1 DYNAMIXEL MX-28T

The MX series is a new set actuators from DYNAMIXEL with advanced functions such as precise control, PID control, 360 degree position control and high speed communication. MX-28T motors are used at the hip roll, pitch and knee pitch DoFs in the structure where the torque requirements reaches 2.5 Nm. It has a 72MHz, 32bit ARM Cortex-M3 microcontroller unit in it to control and coordinate its operation. MX-28T supports a 3 pin TTL half duplex asynchronous serial communication with 8bit, 1 stop, no parity. The MX-28T actuator is shown in figure 5.6, its CAD drawings is shown in figure 5.7 and the specifications are tabulated in table 5.2. To control the DYNAMIXEL actuators, the main controller needs to convert its UART signals to the half duplex type. The recommended circuit diagram for this is shown in figure 5.8.

Figure 5.6: DYNAMIXEL MX-28T

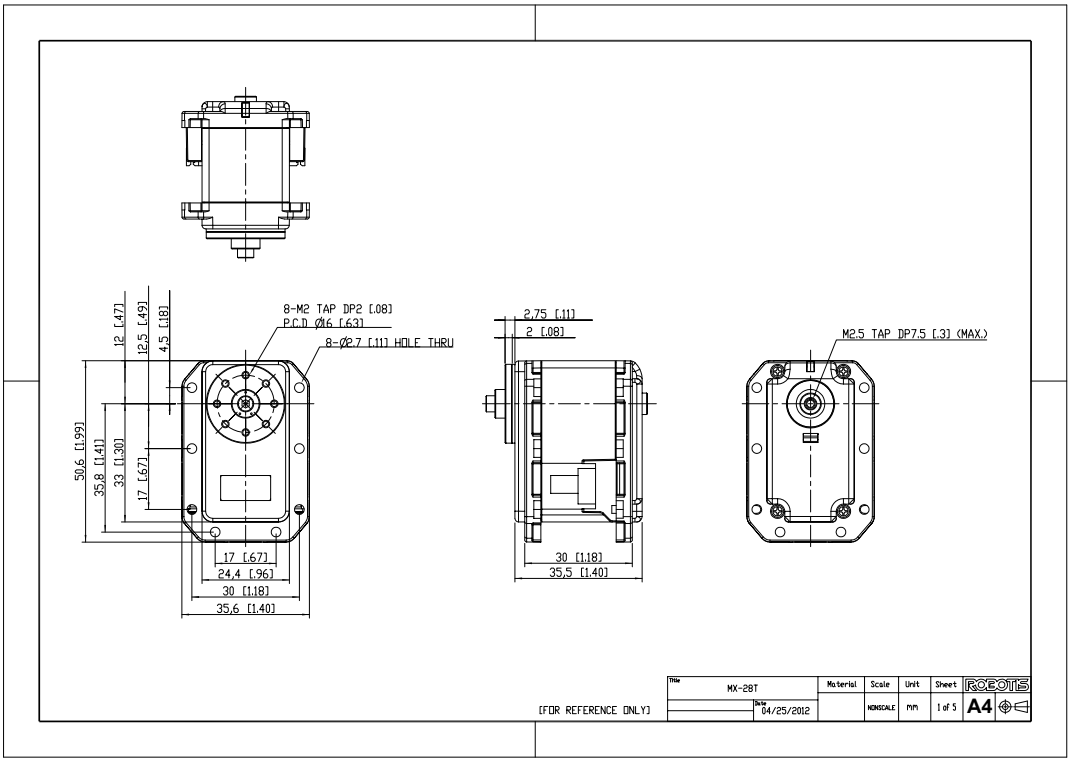

Figure 5.7: MX-28T CAD drawing

Feature	Specification
Baud Rate	8kbps-4.5Mbps
Resolution	0.0879^0
Running Degree	360^0 Endless Turn
Weight (g)	72
Dimensions (WxHxD) (mm)	35.6x50.6x35.5
Gear Ratio	193:1
Stall Torque (Nm)	2.5
No Load Speed(rpm)	55
Operating Temperature(^{0}C)	-5 to 80
Input Voltage (V)	12
Command Signal	Digital Packet
Protocol Type	Half duplex asynchronous serial communication
Physical Connection	TTL level multi drop bus
ID	0-253
Feedback	Position, temperature,load, input voltage

Table 5.2: MX-28T Technical Specifications

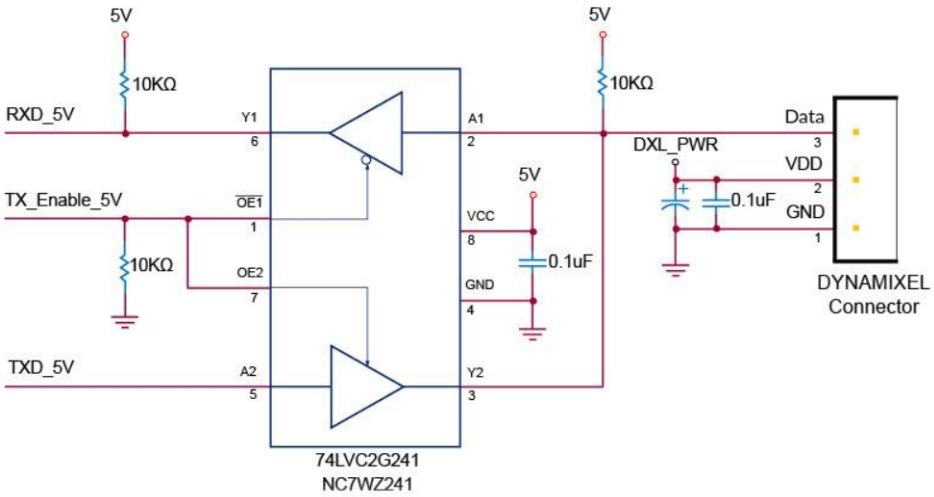

Figure 5.8: TTL connection circuit of MX-28T

5.2.2 DYNAMILXEL AX-18A

Dynamixel AX series is an array of actuators that provide better performance at low cost. AX-18A is a high performance version of the popular Dynamixel AX-12A, and can be applied to robots that need more power. These actuators are used at the ankle pitch and roll DoFs where the torque requirement reaches 1.8 Nm. It supports only 3 pin TTL level half duplex asynchronous serial communication 8 bit, 1 stop, no parity. The Dynamixel AX-18A is shown in figure 5.9 and its CAD drawing is illustrated in figure 5.10. The technical specifications of the model is given in table 5.3. To control the DYNAMIXEL actuators, the main controller needs to convert its UART signals to the half duplex type. The recommended circuit diagram for this is shown in figure 5.11. This TTL communication circuit is designed for 5V tolerant MCU.

Figure 5.9: DYNAMIXEL AX-18A

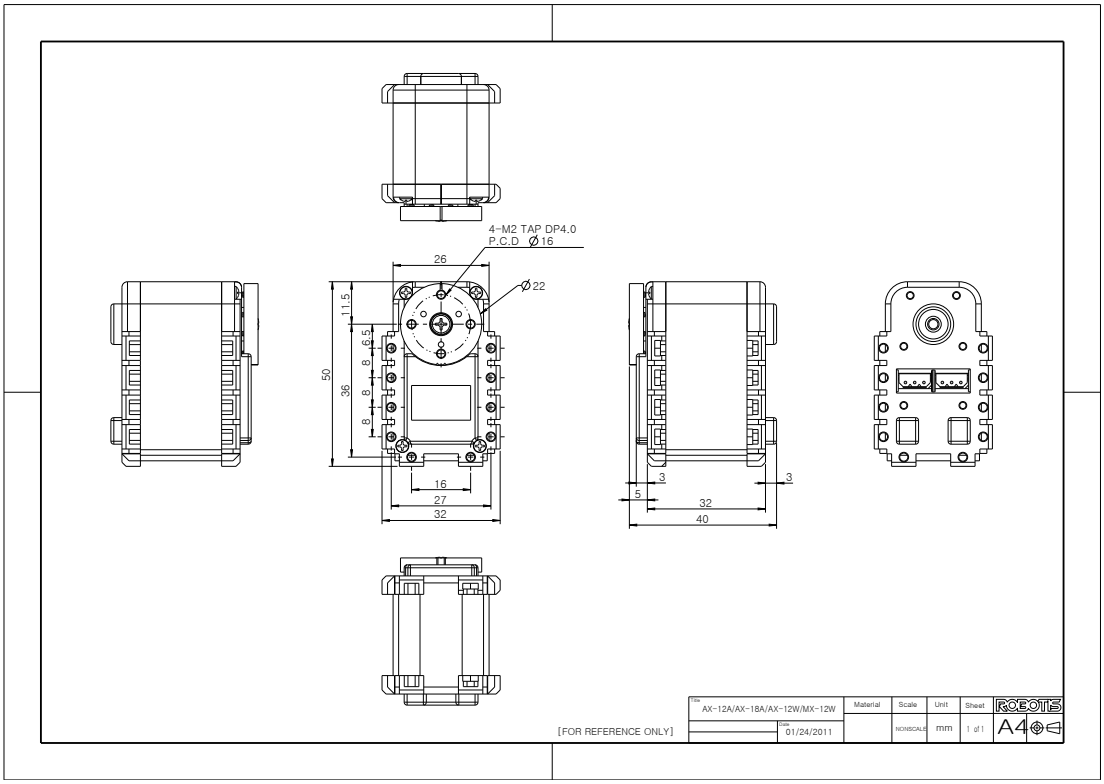

Figure 5.10: AX-18A CAD drawing

Feature	**Specification**
Baud Rate	7843-1Mbps
Resolution	0.293^0
Running Degree	300^0 Endless Turn
Weight (g)	56
Dimensions (WxHxD) (mm)	32x50x40
Gear Ratio	254:1
Stall Torque (Nm)	1.8
No Load Speed(rpm)	97
Operating Temperature(^{0}C)	-5 to70
Input Voltage (V)	12
Command Signal	Digital Packet
Protocol Type	Half duplex asynchronous serial communication
Physical Connection	TTL level multi drop bus
ID	0-253
Feedback	Position, temperature,load, input voltage

Table 5.3: AX-18A Technical Specifications

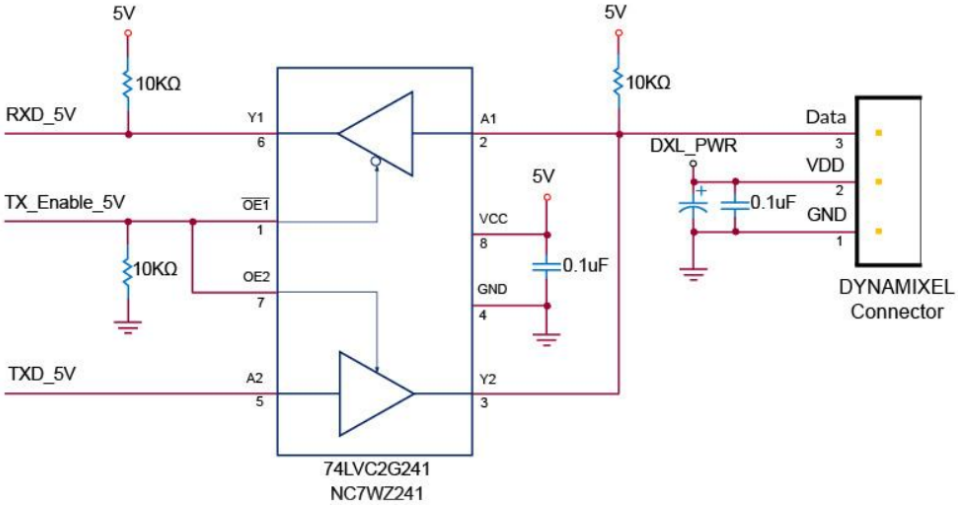

Figure 5.11: TTL connection circuit of AX-18A

5.2.3 DYNAMIXEL AX-12A

Dynamixel AX-12A is another smart actuator from the AX series. It has lesser torque capabilities when compared to the MX series motors and AX-18A motors and hence it is used for the hip yaw motion where the torque requirement is nearly 1 Nm. The constructional features and dimensions are identical to the AX-18A motors. Unlike AX-18A motors, AX-12A motors contain all plastic gears. It supports 3 pin TTL level half duplex asynchronous serial communication 8 bit, 1 stop, no parity. To control the DYNAMIXEL actuators, the main controller needs to convert its UART signals to the half duplex type. The recommended circuit diagram for this is shown in figure 5.14. This circuit is designed for 5V tolerant MCU.

Figure 5.12: DYNAMIXEL AX-12A

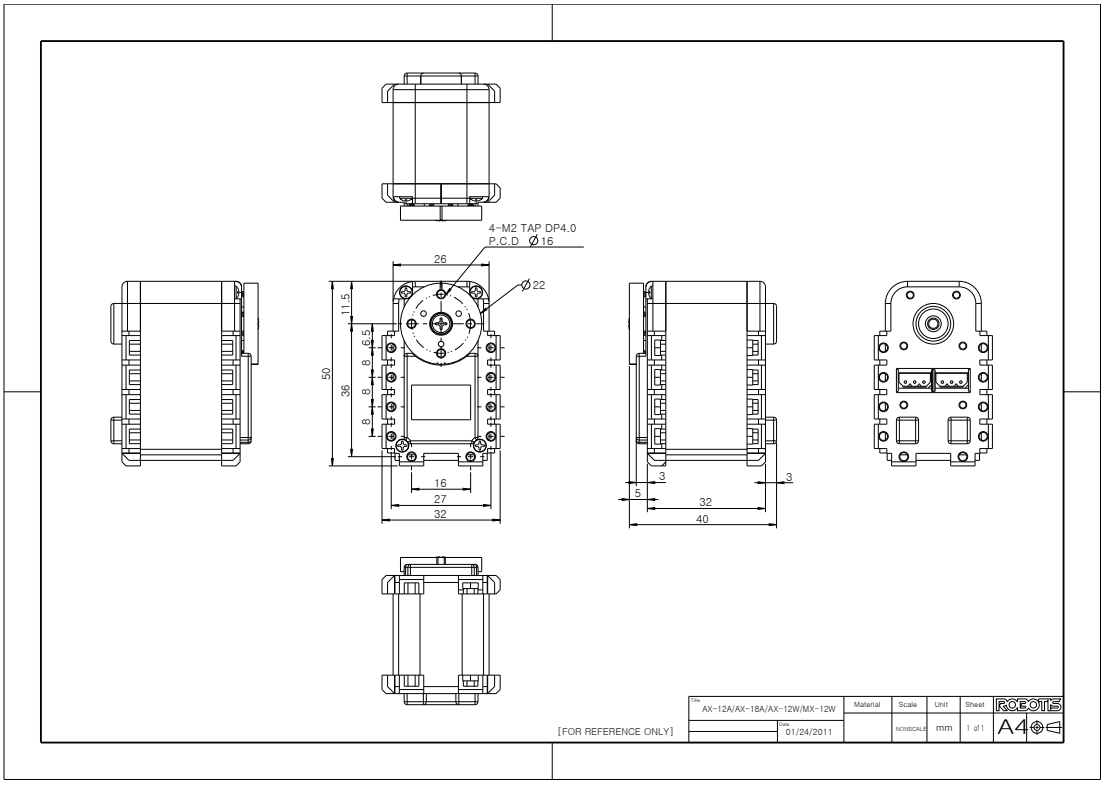

Figure 5.13: AX-12A CAD drawing

Feature	Specification
Baud Rate	7843-1Mbps
Resolution	0.2930^0
Running Degree	300^0 Endless Turn
Weight (g)	56
Dimensions (WxHxD) (mm)	32x50x40
Gear Ratio	254:1
Stall Torque (Nm)	1.5
No Load Speed(rpm)	59
Operating Temperature(^{0}C)	-5 to70
Input Voltage (V)	12
Command Signal	Digital Packet
Protocol Type	Half duplex asynchronous serial communication
Physical Connection	TTL level multi drop bus
ID	0-253
Feedback	Position, temperature,load, input voltage

Table 5.4: AX-12A Technical Specifications

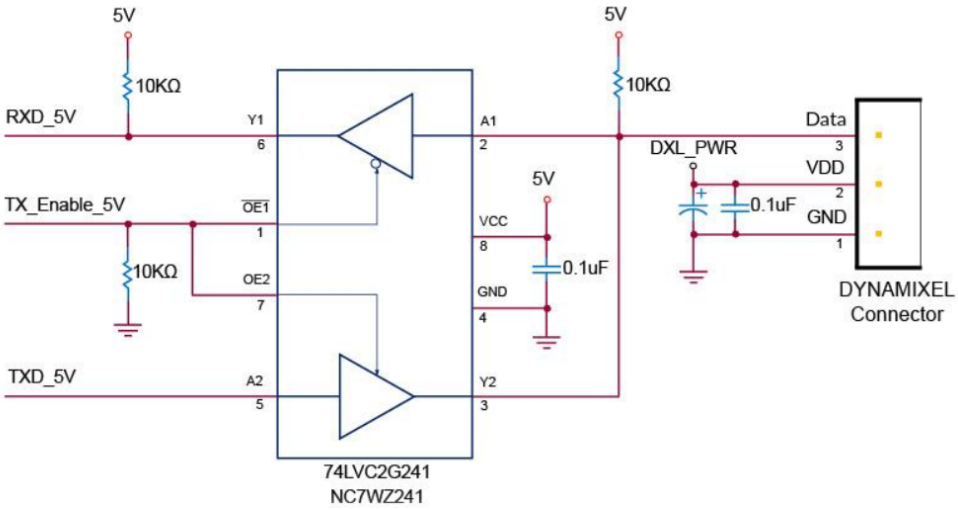

Figure 5.14: TTL connection circuit of AX-12A

5.3 Design of Parts

When verifying walking control of a humanoid robot, even if its effectiveness is confirmed through simulation, the controller often doesn't work with an actual robot. This is mainly due to model errors such as deflection of the actual robot, servo stiffness, and errors of various sensors. It is important to develop balance control that can compensate for the model errors, but it is also important to develop robot hardware that behaves close to an ideal model and that is easy to be modeled from the viewpoint of mechanics. The legs of a humanoid robot are generally designed with attention to the following points:

1. Reduce model error

2. Reduce leg inertia

In order to reduce the model error, each link is designed to have high stiffness against bending and torsion, and a mechanism with small backlash is often adopted for each joint. For reducing the leg inertia, the end link is designed to have a low mass. By taking these two points into consideration, the prototype developed in this work consists of 17 parts in total which are equally distributed between both the legs [10]. The modelling is done in Solid Works.

5.3.1 Hip-Base-Link

This part is designed to be the base link for the entire structure. It is common for both the legs. The structure is 2mm thick and is modelled in Aluminium 6063 Alloy. The actuators for left and right hip roll motion are fixed on either side of this link. The part weighs 24.61 grams with a volume of 9113.79mm^3.

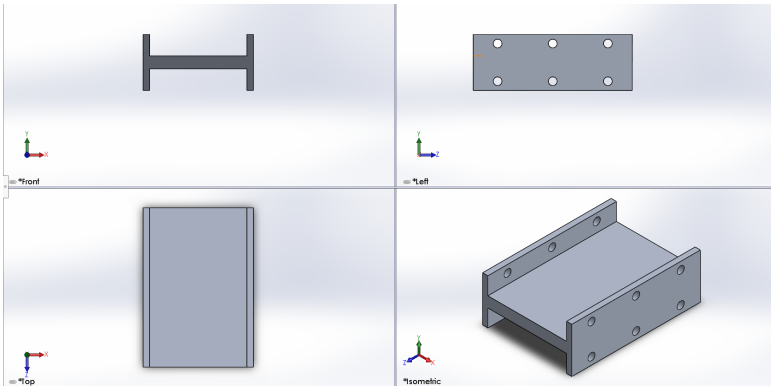

Figure 5.15: Hip base link model

5.3.2 Hip-Clamp-1

This clamp is modelled in such a way that it facilitates the roll motion at the hip roll actuator. It is attached to the hip roll motor head at the circular hole provided on the clamp. The clamp weighs 14.74grams and has a volume of 5460.69mm^3. The material used is Aluminium 6063 Alloy.

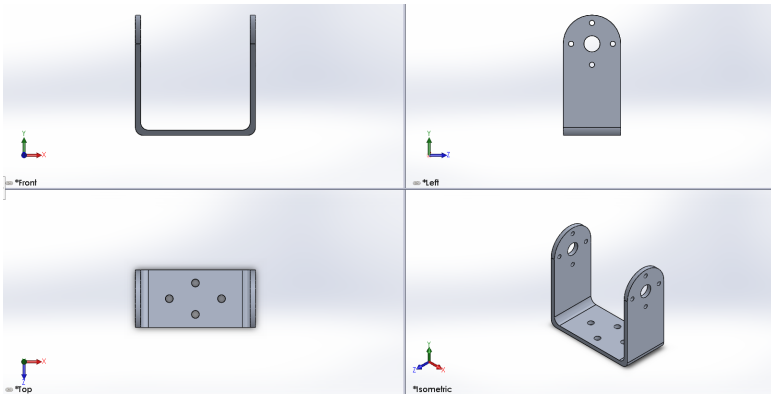

Figure 5.16: Hip Clamp 1

5.3.3 Hip-Clamp-2

This part doesn't facilitate any motion to the structure, instead it is used to attach the Hip yaw motion actuator to the structure at the hip clamp 1. It is attached to the rear end of the hip clamp 1 at the 4 circular holes using screws and the yaw motion actuator is attached to this hip clamp 2. It is fabricated in Aluminium 6063 Alloy and weighs 13.37grams and has a volume of 4952.4 mm^3.

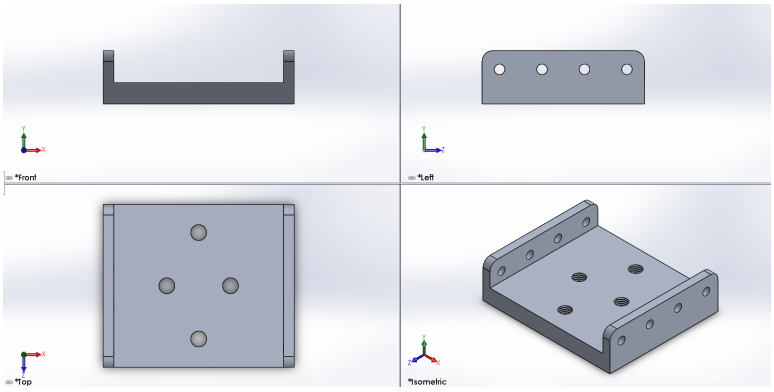

Figure 5.17: Hip Clamp 2

5.3.4 Hip-Clamp-3

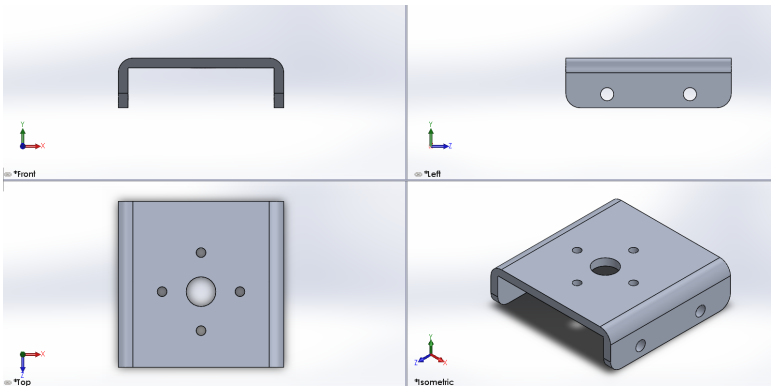

Figure 5.18: Hip Clamp 3

This clamp is designed based on the motor head dimensions of the hip yaw actuator and hence provides the yaw motion at the hip. It is attached to the yaw motion actuator motor head at the circular holes provided. Also, the rear end of the Hip pitch actuator is bolted to this clamp at the 4 circular holes along the projecting part of the clamp. It doesn't provide any relative motion to this pitch motion actuator. The structure weighs 8.51grams and has a volume of 3151.03mm^3. The material used is Aluminium 6063 Alloy.

5.3.5 Thigh-Link

This link represents the thigh of the robot and has dimensions as per the anthropometric data mentioned in section 5.1. It's upper end is attached to

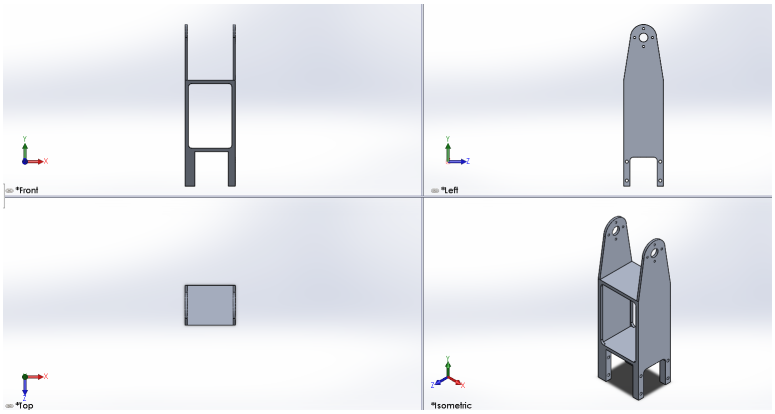

Figure 5.19: Thigh Link model

the motor head of the hip pitch actuator at the circular holes provided in the link and the lower end is used for accommodating the knee pitch actuator without any relative motion at the knee. The design of the part is done carefully by reducing the mass and weight without compromising with the structural rigidity and stability. The link weighs around 81.18grams and has a volume of 30066.21 mm^3. Material used for fabrication is Aluminium 6063 alloy.

5.3.6 Calf-Link

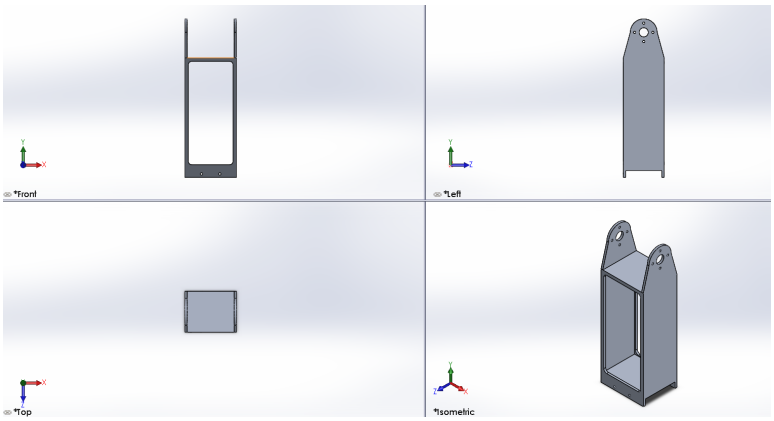

Figure 5.20: Calf Link model

This link represents the calf of the robot and has dimensions as per the anthropometric data mentioned in section 5.1. It's upper end is attached to the motor head of the knee pitch actuator at the circular holes provided in the

link and the lower end is used for accommodating the ankle roll actuator without any relative motion at the ankle. The design of the part is done carefully by reducing the mass and weight without compromising with the structural rigidity and stability. The link weighs around 79.35grams and has a volume of 29388.26 mm^3. Material used for fabrication is Aluminium 6063 alloy.

5.3.7 Ankle-Clamp-1

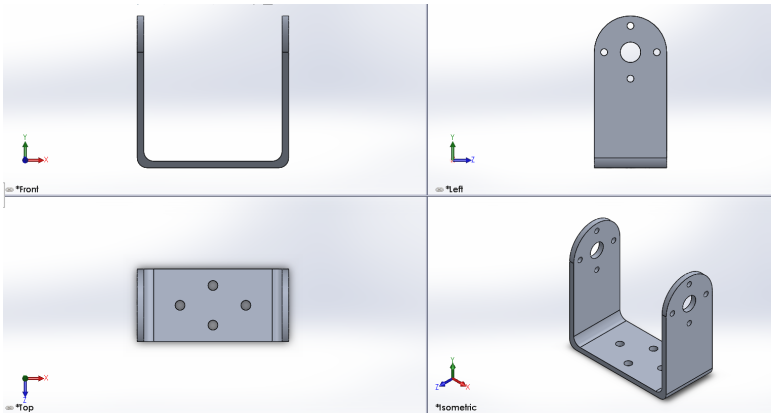

Figure 5.21: Ankle Clamp 1

This part is designed in such a way it provides the roll motion at the ankle roll actuator. The ankle roll actuator motor head is attached directly to this clamp at the circular holes provided. The rear end of the clamp is attached to the ankle clamp 2 without any relative motion. The clamp weighs 14.73grams and has a volume of 5455.54mm^3. Material used for fabrication is Aluminium 6063 alloy.

5.3.8 Ankle-Clamp-2

This part doesn't facilitate any motion to the structure, instead it is used to attach the Ankle pitch motion actuator to the structure at the ankle clamp 1. It is attached to the rear end of the ankle clamp 1 at the 4 circular holes using screws and the pitch motion actuator is attached to this ankle clamp 2. It is fabricated in Aluminium 6063 Alloy and weighs 13.46grams and has a volume of

4984.39 mm^3.

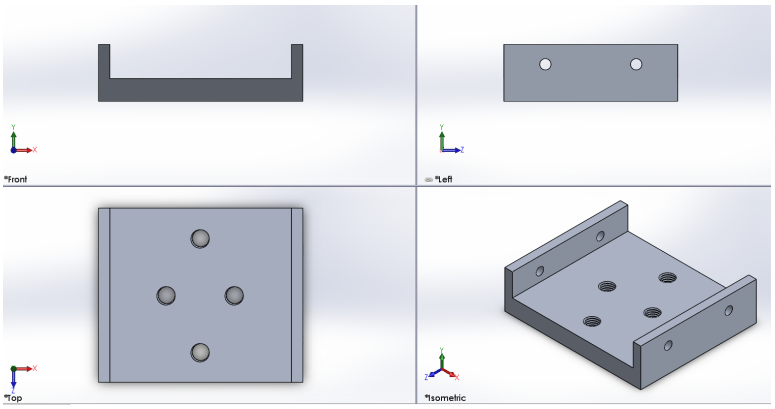

Figure 5.22: Ankle Clamp 2

5.3.9 Foot Link

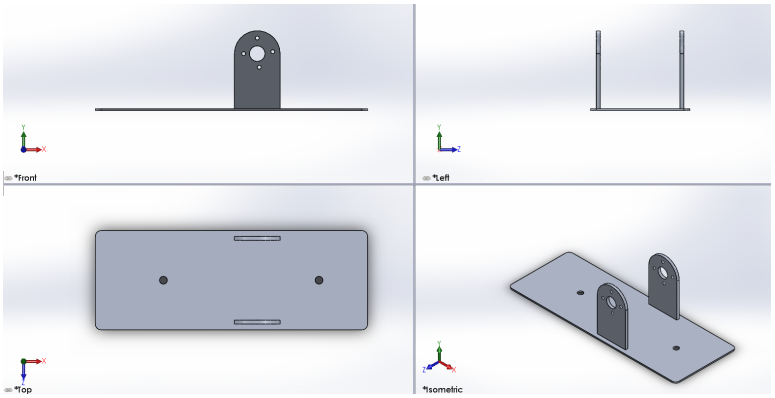

Figure 5.23: Foot Link model

This link represents the foot of the robot and has dimensions as per the anthropometric data mentioned in section 5.1. It's upper end is attached to the motor head of the Ankle pitch actuator at the circular holes provided in the link and the lower end is used for stepping over the surface of the external environment. The design of the part is done carefully by reducing the mass and weight without compromising with the structural rigidity and stability. The link weighs around 47.35grams and has a volume of 17536.19 mm^3. Material used for fabrication is Aluminium 6063 alloy.

5.4 Hardware Interface

The Structure to PC interfacing is done using a small size USB communication converter named U2D2. It enables to control and operate DYNAMIXEL with PC. U2D2 can be connected to the USB port of the PC with the enclosed USB cable. It supports both 3Pin TTL connector and 4Pin RS-485 connector to link up with various DYNAMIXEL's. U2D2 does not supply power to DYNAMIXEL, therefore, an external power supply should provide power to DYNAMIXEL. The layout of U2D2 is shown in figure 5.24 and the specifications are shown in table 5.5. The PC to DYNAMIXEL interfacing using U2D2 is illustrated using figure 5.25.

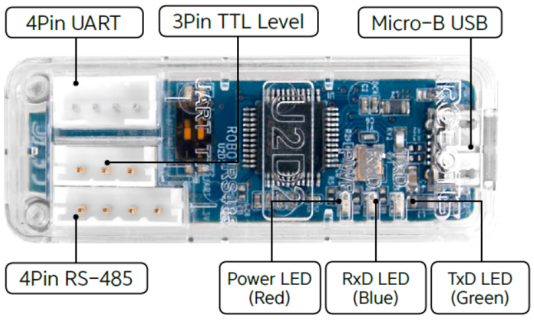

Figure 5.24: U2D2 Layout

Feature	Description
Weight(g)	9
Dimensions(mm)	48x18x14.6
Ports	3 pin TTL level, 4 pin RS-485, 4 pin UART
Baud Rate	Maximum 6Mbps

Table 5.5: U2D2 Specifications

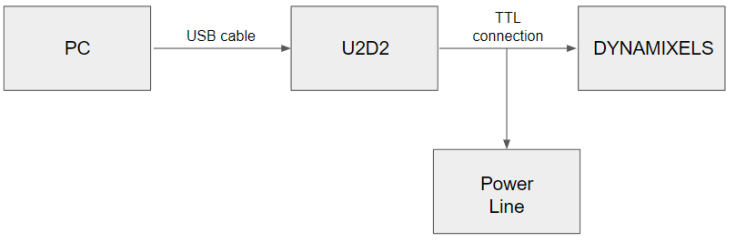

Figure 5.25: PC to DYNAMIXEL interfacing

5.5 Power Source

The power requirements of the robot is met using a Switched Mode Power Supply (SMPS). It is an electronic circuit that converts power using switching devices that are turned on and off at high frequencies, and storage components such as inductors or capacitors to supply power when the switching device is in its non-conduction state. A 12V, 10A SMPS was used to carry out the testing of the robot.

Figure 5.26: 12V, 10A SMPS

5.6 Developed Prototype

A CAD model of the prototype with six degrees of freedom for each leg was developed in Solid Works. The dimensional accuracy and strength analysis for different parts were carried out. Then the individual parts were fabricated using Aluminium Fabrication with Aluminum 6063 Alloy. The fabricated parts along with motors were assembled accordingly. The Dynamixel actuators were connected as per the respective connection diagrams. SMPS and other control boards were mounted on the assembled model. Thus a prototype as shown in figure 5.27 was evolved.

Hence, the primitive model design was implemented in Simscape Multibody and the torque outputs of each joint analysed. Based on the torque output, suitable actuators were selected for each joint and thereafter detailed design of

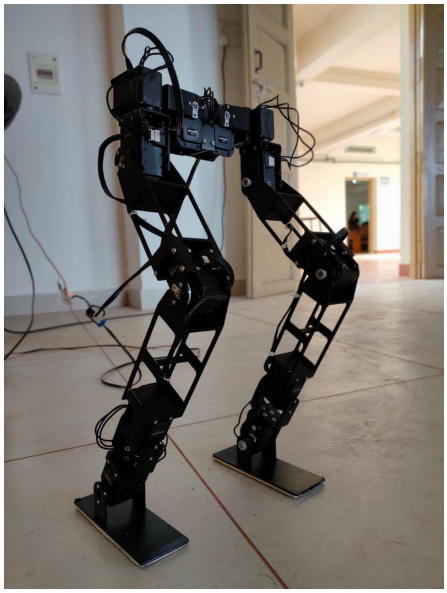

Figure 5.27: Developed hardware prototype

the parts were done in SolidWorks. Designed parts were fabricated in Aluminium 6063 alloy and structure was assembled. The software design for the control of the hardware is discussed in the next chapter.

Chapter 6

Software Design

Inorder to control the hardware model, the design of an effective control algorithm is required. Some of the softwares the we have used include, Solidworks, Matlab and Simulink,ROSand Dynamixel Wizard. The CAD modeling of the prototype is done using Solidworks, which can be used for developing the control algorithm for static stability in standing posture. Here come across various steps including the URDF generation, methods to establish the standing posture, interfacig to the hardware with ROS, develpoing the ROS subscriber and publisher nodes and associatig Matlab with ROS.

6.1 Structure Design

The structural design was initiated with a primitive model as mentioned in section 5.1. The actuators for each joint were selected based on the performance of the primitive model and thereafter the parts of the actual model were designed as per section 5.3. These parts were assembled suitably along with the CAD models of the dynamixel motors and the coordinate system for each part and motor were defined. The Hip-Base-Link was assigned the base coordinate system for the entire structure and finally an assembly file of the complete model (figure 6.1) was generated in SolidWorks. Each joint motion were assigned maximum and

minimum angle limit in degrees as shown in table 6.1. The joint angle limits were set by considering the collision of the parts among each other.

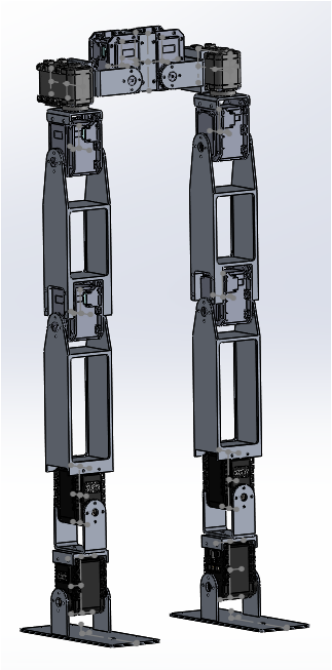

Figure 6.1: SolidWorks model of the structure

Joint	**Angle Limits (degrees)**
Hip Roll	−40 to 0 to +120
Hip Pitch	−130 to 0 to +120
Hip Yaw	−150 to 0 to +150
Knee Pitch	−50 to 0 to +110
Ankle Roll	−120 to 0 to +120
Ankle Pitch	−100 to 0 to +90

Table 6.1: Joint angle limits

6.1.1 URDF Generation

URDF(Unified Robotics Description Format) is an XML(extensible Markup Language) format for representing a robot model. It represents the appearance of the robot and its intended action. It is used to generate robotic models in various commercial simulators such as Gazebo. In the URDF file, both links and joints are described separately. These descriptions are provided as XML

elements such as $< link >$, $< joint >$ etc nested in hierarchical structures known as XML trees. The link structure comprises of child elements including $< name >, < inertial >, < visual >$ etc that defines the name, properties and behavior of the particular link. Similarly, the joint structure comprises of child elements including $< name >, < parentlink >, < childlink >$ etc that defines the name, the previous link to which the joint is attached and the succeeding link that follows the joint. The URDF of the model was imported to matlab and the rigid body tree structure of the model was obtained as shown in figure 6.2.

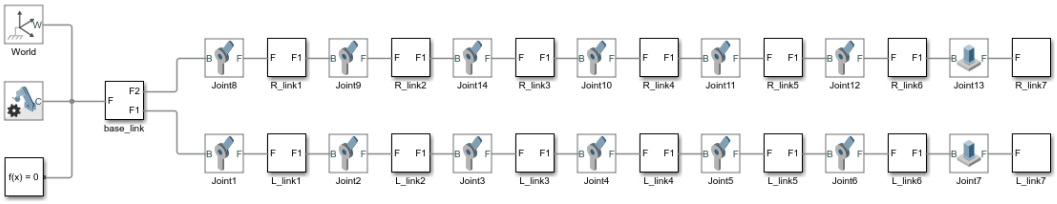

Figure 6.2: Rigid body tree of the structure obtained from the URDF

6.2 Establishing Standing Posture

For a biped robot, stability in standing posture is established when the projection of the COM falls within the support polygon of the model. There can be different configurations in which this constraint can be satisfied. For each configuration, the position of the COM is first identified and based on this the joint angles are calculated using inverse kinematics solver. Further, these angles are sent to the model. The algorithm used for this function is included in Appendix A.

Steps involved in the algorithm:

1. Inverse kinematics model of the robot is developed based on the position of the COM and feet.

2. Based on priority of joint actuations, proper weightages are provided for each DoF.

3. The inverse kinematics model calculates the joint angles based on the foot position, COM position and the weights assigned for each joint.

6.3 ROS - Hardware Interfacing

ROS stands for "Robot Operating System", but it is not actually an operating system. ROS is actually a set of software libraries and tools made to ease the development of robotic applications. It provides the services expected from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. ROS is widely used in designing, building, and simulating a robot model and interfacing it into real hardware.

- **Ros node:** A ROS node is an executable program which contains a part of code of the robot. A robotic application can have more than one node. There is a Master node running which will have all the address of the nodes running in the system. It is Master node's responsibility to connect the Nodes with each other to start communication.

- **ROS message:** ROS message provides the data structure for Topics, Service and Actions to transfer information between nodes.

- **ROS topic:** Topics are named buses over which nodes exchange messages. This is the most commonly used mode of communication for nodes in ROS.

- **Publisher node:** A node broadcast a message with a topic name. This broadcasting node is known as publisher node.

- **Subscriber node:** A node that want to receive the message will subscribe to this topic name. This node is known as subscriber node.

6.3.1 Motor Control Table Setup

The Control Table is a structure that consists of multiple Data fields to store status or to control the device. We can check current status of the device by reading a specific Data from the Control Table with Read Instruction Packets. WRITE Instruction Packets enable users to control the device by changing specific Data in the Control Table. In order to read or write data, we must designate a specific Address in the Instruction Packet. With the help of proper Address and parameters we can control the actuators. Control table of EEPROM and RAM area of the Dynamixel actuators are shown in figure 6.3.

Address	Size(Byte)	Data Name	Description	Access	Initial Value
0	2	Model Number	Model Number	R	30
2	4	Model Information	Model Information	R	-
6	1	Firmware Version	Firmware Version	R	-
7	1	ID	DYNAMIXEL ID	RW	1
8	1	Baud Rate	Communication Baud Rate	RW	1
9	1	Return Delay Time	Response Delay Time	RW	250
10	1	Drive Mode	Drive Mode	RW	0
11	1	Operating Mode	Operating Mode	RW	3
12	1	Secondary(Shadow) ID	Secondary ID	RW	255
13	1	Protocol Type	Protocol Type	RW	2
20	4	Homing Offset	Home Position Offset	RW	0
24	4	Moving Threshold	Velocity Threshold for Movement Detection	RW	10
31	1	Temperature Limit	Maximum Internal Temperature Limit	RW	80
32	2	Max Voltage Limit	Maximum Input Voltage Limit	RW	160
34	2	Min Voltage Limit	Minimum Input Voltage Limit	RW	95
36	2	PWM Limit	Maximum PWM Limit	RW	885
40	4	Acceleration Limit	Maximum Acceleration Limit	RW	32767

Figure 6.3: Control Table of Dynamixel actuators

6.3.2 ROS Subscriber node

The ROS Subscriber Node is used to send the angle values to the motors. It is used to establish a serial communication with the motors, set proper baud rates and send the parameters. The control table parameters required for writing the subscriber node are Address torque enable, Address present position and Address goal position. Here the joint angle values obtained from the controller is converted into 10 bit binary data packets and sent to the respective motors. Connection with the motors is established when the motor ID, serial port and baud rate are tallied. Once connection is established, the torque enable bit is set high and the in Appendix B.

6.3.3 ROS Publisher node

A ROS publisher node is used to obtain the angles from the MATLAB and send it to the subscriber node via the set goal position topic. The input to the publisher node is the motor ID and its corresponding angle. The simulink block diagram of the publisher node is shown in figure 6.4.

6.3.4 MATLAB - ROS interfacing

The angle values obtained from the MATLAB script is imported into the workspace in time table format. From the workspace, these values are imported into the Simulink block diagram of the publisher node. Using a mapping function, the angles falling in the range -150 to 0 to +150 is converted into the range of 0 to 1023. These converted values are passed on to the publishing node.

Once the Matlab is connected to ROS, the hardware interfacing is completed. Hence we can move on the hardware testing and conclusions.

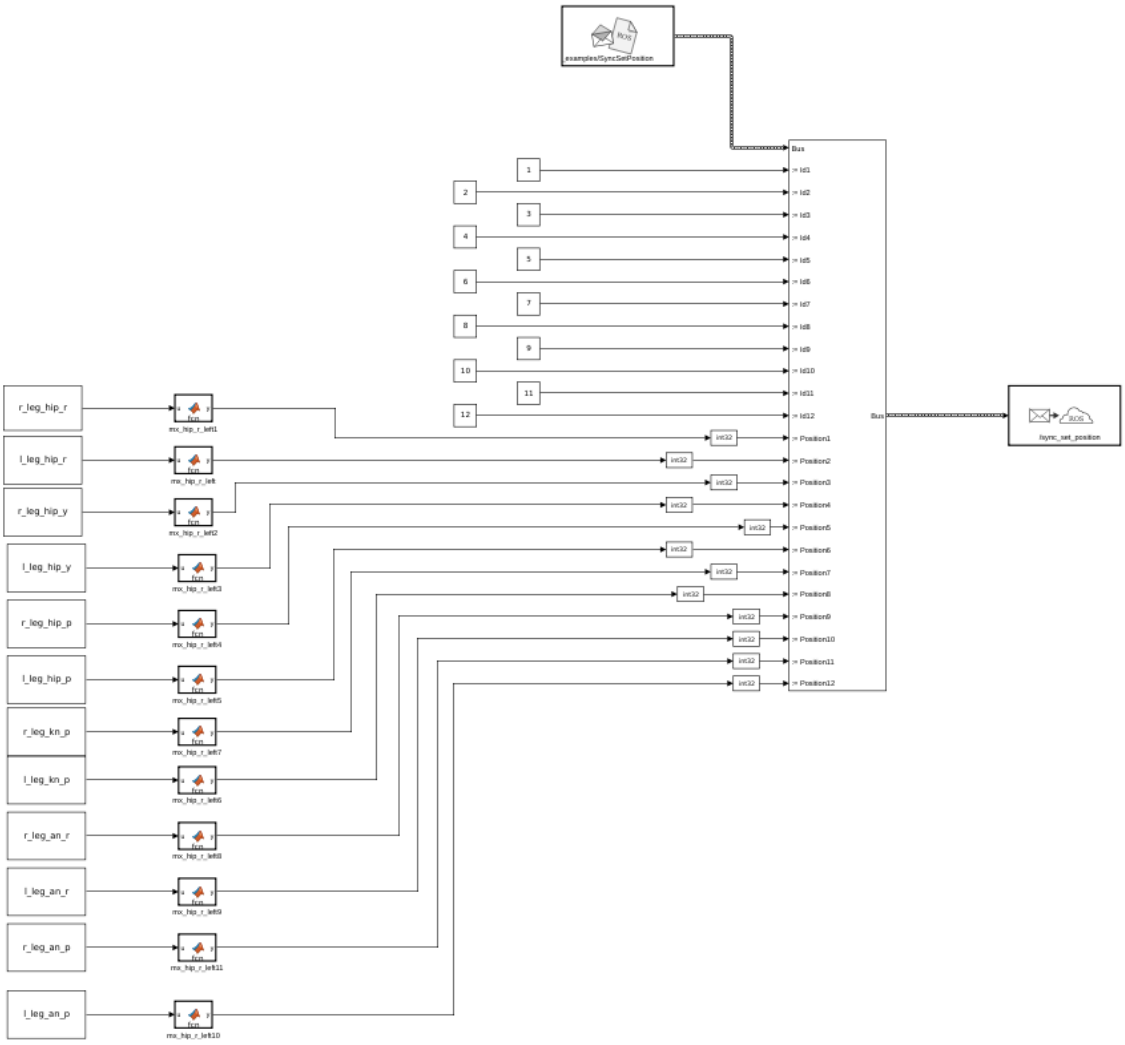

Figure 6.4: Simulink Block diagram to provide joint angles to ROS

Chapter 7

Results and Discussions

The final structure is designed based on the anthropometric data and the primitive model. SolidWorks model of the robot is designed and fabricated in aluminium 6063 alloy. The results of the algorithm for stability in the standing posture and implementation of the walking gait in the hardware model is discussed in this chapter.

7.1 Primitive Model

The height of the model was fixed to be 50cm and the link dimensions were selected based on the anthropometric data. A primitive model of the robot was designed with the 12 DOF joint distribution and the calculated link dimensions in Simscape Multibody as shown in figure 7.1.

7.2 Actual Model

The CAD model of the complete structure was developed in Solid Works. Dynamic simulation of the model was done in gazebo with ROS. Static stability for standing posture was established in gazebo with the projection of COM falling within the support polygon as illustrated in figure 7.4. The most stable posture

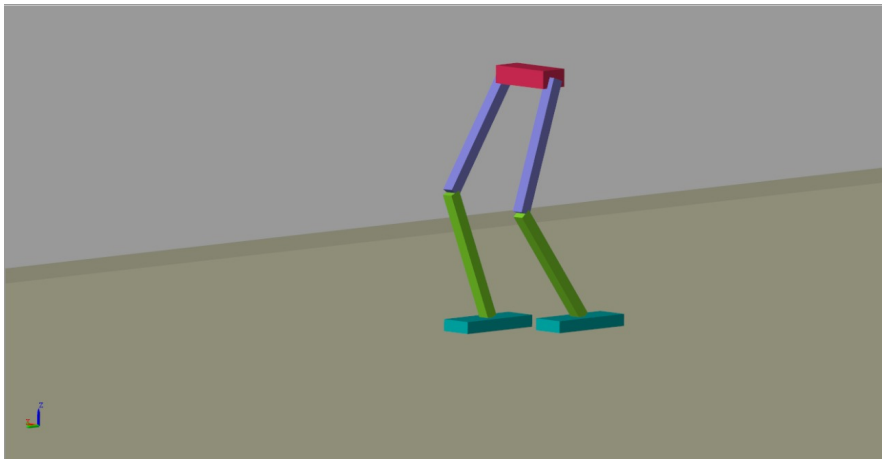

Figure 7.1: Primitive Model

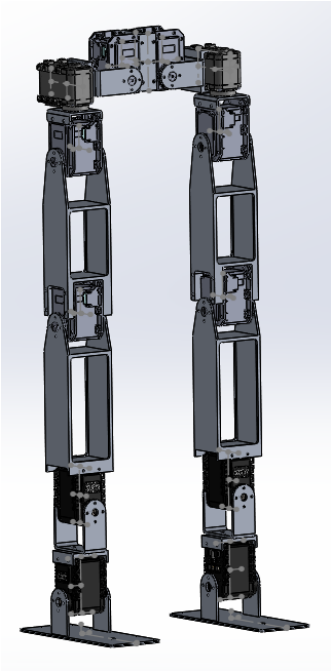

Figure 7.2: CAD model of the structure

for static stability was obtained when the COM of the structure was at a height of 38cm from the ground. From figure 7.3, it is evident that this stable posture represents a crouching position of the robot.

The joint angles for establishing this crouching posture were also plotted (figure: 7.5) and it was clear that the values of the angles were falling within the maximum and minimum joint angle limits specified.

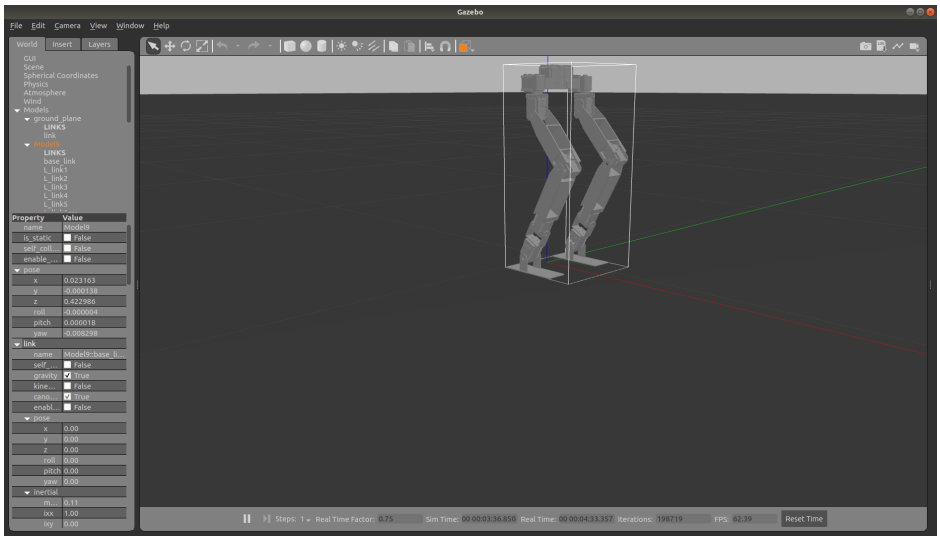

Figure 7.3: Crouching posture for stability

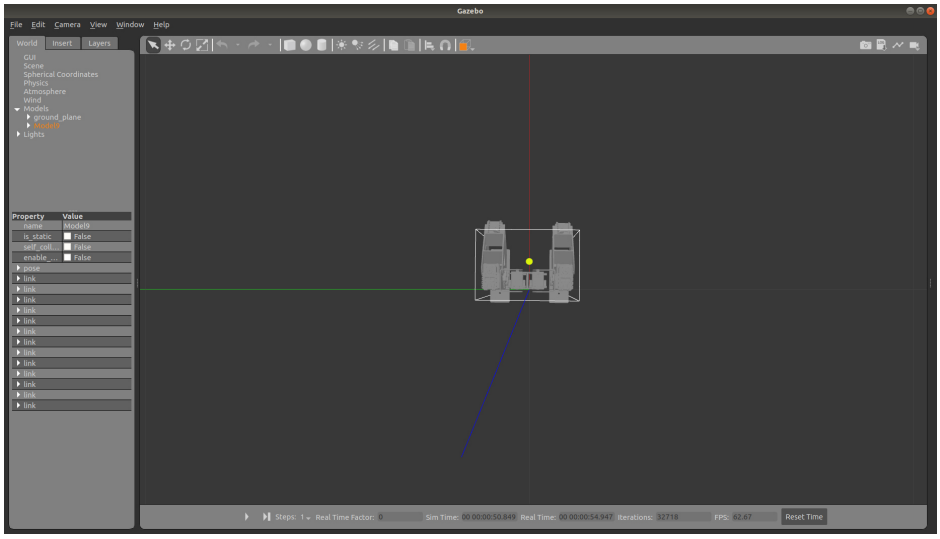

Figure 7.4: COM and Support Polygon

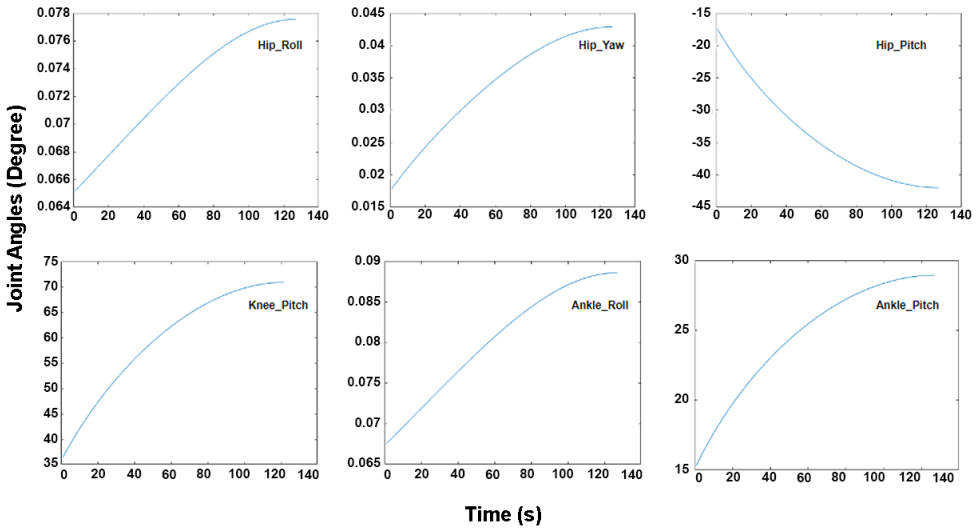

Figure 7.5: Joint Angles for standing posture

7.3 Developed Prototype

The overall system was designed and implemented structurally. The developed hardware was tested by conducting experiments on a smooth surface. Inorder to obtain static stability, the COM projection must fall within the support polygon. Figure 7.4 represents the gazebo simulation result wherein this constraint is met. For meeting this constraint, the COM had to be shifted down to a height of 38cm and hence the robot acquires a crouching position as shown in figure 7.3. The joint angles obtained for this standing posture is plotted in figure 7.5 and was supplied to the hardware model and the result was obtained as shown in figure 7.6.

Implementation of walking motion in the robot was also successfully accomplished with a set of suitable time stamped joint angles as shown in figure 7.7.

These joint angles generated is tested in the actual hardware model by placing it on a plane surface (Figure 7.8). At first robot moves to crouching position from steady standing level. Then it starts moving each of the actuators so as to enable walking. Slight adjustments in the parameter values are made to overcome the

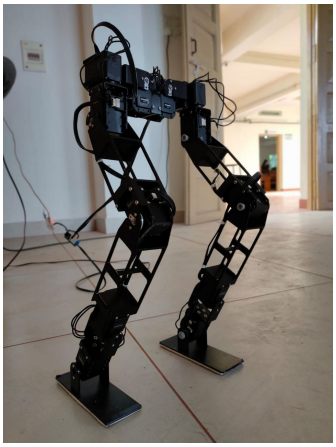

Figure 7.6: Crouching position - static stability in standing posture

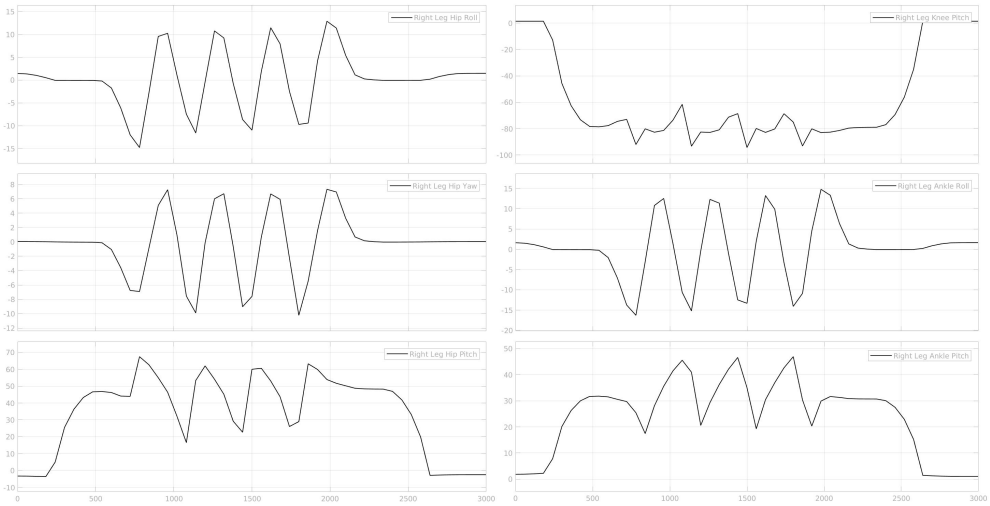

Figure 7.7: Joint Angles for Biped walking

natural effects of friction, gravity, delay in passing angle values and other external conditions. A very small external support has to be given to the hip of the robot from falling. This is due to the absence of torso or upper body for the biped robot to counteract the reactive forces during walking. This can be considered for the future expansion of the proposed Humanoid Biped Robot.

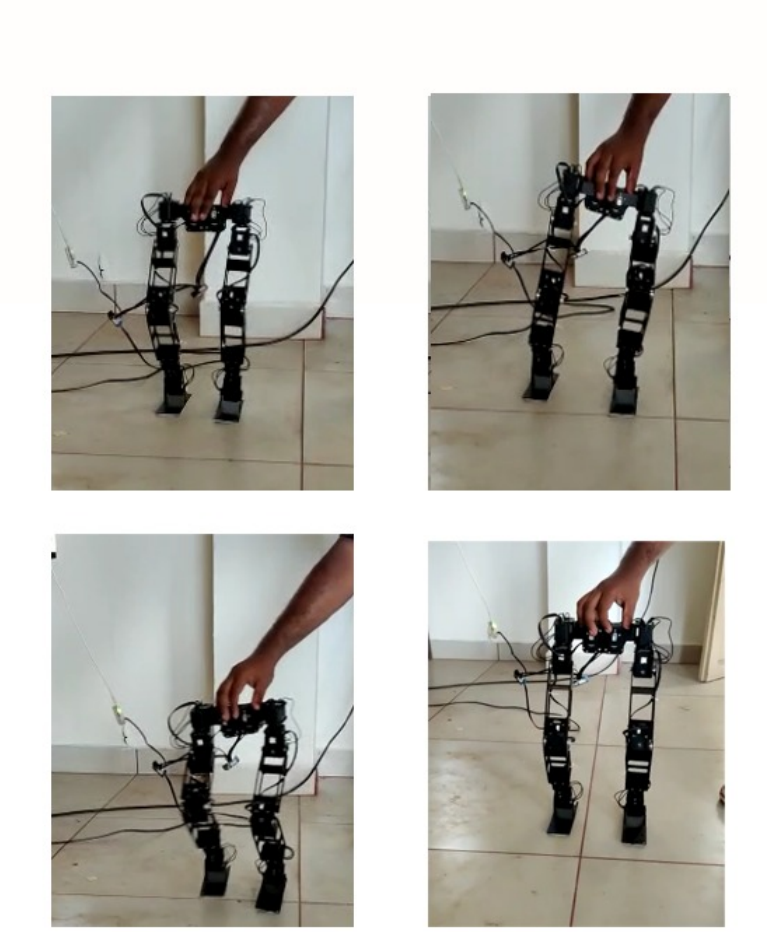

Figure 7.8: Biped walking established in the hardware model

Chapter 8

Conclusion and Future Scope

The goal of this project work was to design a human like robot's lower body suitable for humanoid walking. The model is anthropometric as the configuration and link dimensions are comparable to human beings. A primitive model of the proposed model is designed in Simscape Multibody and the torque requirement of each joint is analysed carefully. Based on the torque requirements, suitable smart actuators are selected from the wide range of Dynamixel actuator options.

The actual model is developed in SolidWorks and URDF file is generated. Also, the detailed design of the different parts is finalised, simulated and fabricated for hardware implementation. Hardware implementation of the model is completed and static stability in standing pose is tuned and verified. The developed structure was also tested with joint angles for biped walking and a stable walking gait was achieved successfully.

References

[1] prof. dr. H. Nijmeijer, prof. dr. ir. P.P. Jonker, dr. ir. P.C.J.N. Rosielle, and ir. E. Dekkers, *Mechanical design of a humanoid robot's lower body.* Eindhoven University of Technology, the Netherlands: Master's Thesis, 2010.

[2] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of honda humanoid robot," in *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, vol. 2, 1998, pp. 1321–1326 vol.2.

[3] Y. Sakagami, R. Watanabe, C. Aoyama, S. Matsunaga, N. Higaki, and K. Fujimura, "The intelligent asimo: system overview and integration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2478–2483 vol.3.

[4] K. Nishiwaki, T. Sugihara, S. Kagami, F. Kanehiro, M. Inaba, and H. Inoue, "Design and development of research platform for perception-action integration in humanoid robot: H6," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 3, 2000, pp. 1559–1564 vol.3.

[5] K. Kaneko, F. Kanehiro, S. Kajita, K. Yokoyama, K. Akachi, T. Kawasaki, S. Ota, and T. Isozumi, "Design of prototype humanoid robotics platform for hrp," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, 2002, pp. 2431–2436 vol.3.

[6] M. Gienger, K. Loffler, and F. Pfeiffer, "Towards the design of a biped jogging robot," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 4, 2001, pp. 4140–4145 vol.4.

[7] ——, "Walking control of a biped robot based on inertial measurement," in *Proc. of Int. Workshop. on Humanoid and Human Friendly Robotics*, 2002, pp. 22–30.

[8] W. Vukobratovic and B. A. Borovac, "Zero-moment point - thirty five years of its life." *International Jouranl of Humanoid Robotics*, pp. 157–173, 2004.

[9] D. Kulić, *Human Motion Imitation.* Dordrecht: Springer Netherlands, 2019, pp. 1657–1677. [Online]. Available: https://doi.org/10.1007/978-94-007-6046-$2_3$4

[10] C. Hernández-Santos, E. Rodriguez-Leal, R. Soto, and J. Gordillo, "Kinematics and dynamics of a new 16 dof humanoid biped robot with active toe joint," *International Journal of Advanced Robotic Systems*, vol. 9, no. 5, p. 190, 2012.

Appendix A

MATLAB Code for Standing Posture

```
l_tform = trvec2tform(l_fs_com);
r_tform = trvec2tform(r_fs_com);

ik = inverseKinematics('RigidBodyTree',robot);
weights = [0.25 0.25 0.25 1 1 1];
guess = robot.homeConfiguration;
solution = [];

for i=1:size(l_fs_com)
    [l_Soln,l_solnInfo] = ik('L_link7',l_tform(:,:,i),weights,guess);

    [r_Soln,r_solnInfo] = ik('R_link7',r_tform(:,:,i),weights,guess);

    guess = [l_Soln(1,1:6) r_Soln(1,7:end)];
    solution = [solution; guess];
```

Appendix B

ROS Subscriber Node

```
#include <ros/ros.h>
#include "std_msgs/String.h"
#include "dynamixel_sdk_examples/SyncGetPosition.h"
#include "dynamixel_sdk_examples/SyncSetPosition.h"
#include "dynamixel_sdk/dynamixel_sdk.h"

using namespace dynamixel;
//Change the control tables, protocol, baudrate and motor_id accordingly.
// Control table address
#define ADDR_TORQUE_ENABLE    24
#define ADDR_PRESENT_POSITION 36
#define ADDR_GOAL_POSITION    30

// Protocol version
#define PROTOCOL_VERSION      1.0             // Default Protocol version of DYNAMIXEL X series.

// Default setting
#define DXL1_ID               1               // DXL1 ID
#define DXL2_ID               2               // DXL2 ID
#define DXL3_ID               3               // DXL3 ID
#define DXL4_ID               4               // DXL4 ID
#define DXL5_ID               5               // DXL5 ID
#define DXL6_ID               6               // DXL6 ID
#define DXL7_ID               7               // DXL7 ID
#define DXL8_ID               8               // DXL8 ID
#define DXL9_ID               9               // DXL9 ID
#define DXL10_ID              10              // DXL10 ID
#define DXL11_ID              11              // DXL11 ID
#define DXL12_ID              12              // DXL12 ID

#define BAUDRATE              1000000           // Default Baudrate of DYNAMIXEL X series
#define DEVICE_NAME           "/dev/ttyUSB0"  // [Linux] To find assigned port, use "$ ls /dev/ttyUSB*" command

PortHandler * portHandler = PortHandler::getPortHandler(DEVICE_NAME);
PacketHandler * packetHandler = PacketHandler::getPacketHandler(PROTOCOL_VERSION);

GroupSyncRead groupSyncRead(portHandler, packetHandler, ADDR_PRESENT_POSITION, 4);
GroupSyncWrite groupSyncWrite(portHandler, packetHandler, ADDR_GOAL_POSITION, 4);
```

```
bool syncGetPresentPositionCallback(
  dynamixel_sdk_examples::SyncGetPosition::Request & req,
  dynamixel_sdk_examples::SyncGetPosition::Response & res)
{
  uint8_t dxl_error = 0;
  int dxl_comm_result = COMM_TX_FAIL;
  int dxl_addparam_result = false;

  // Position Value of X series is 4 byte data. For AX & MX(1.0) use 2 byte data(int16_t) for the Position Value.
  int16_t position1 = 0;
  int16_t position2 = 0;
  int16_t position3 = 0;
  int16_t position4 = 0;
  int16_t position5 = 0;
  int16_t position6 = 0;
  int16_t position7 = 0;
  int16_t position8 = 0;
  int16_t position9 = 0;
  int16_t position10 = 0;
  int16_t position11 = 0;
  int16_t position12 = 0;


  // Read Present Position (length : 4 bytes) and Convert uint32 -> int32
  // When reading 2 byte data from AX / MX(1.0), use read2ByteTxRx() instead.

  //Dynamixel_1
  dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id1);
  if (dxl_addparam_result != true) {
    ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id1);
    return 0;
  }

  //Dynamixel_2
  dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id2);
  if (dxl_addparam_result != true) {
    ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id2);
    return 0;
  }
//Dynamixel_3
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id3);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id3);
  return 0;
}

//Dynamixel_4
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id4);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id4);
  return 0;
}

//Dynamixel_5
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id5);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id5);
  return 0;
}

//Dynamixel_6
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id6);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id6);
  return 0;
}

//Dynamixel_7
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id7);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id7);
  return 0;
}
```

```
//Dynamixel_8
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id8);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id8);
  return 0;
}

//Dynamixel_9
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id9);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id9);
  return 0;
}

//Dynamixel_10
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id10);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id10);
  return 0;
}

//Dynamixel_11
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id11);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id11);
  return 0;
}

//Dynamixel_12
dxl_addparam_result = groupSyncRead.addParam((uint8_t)req.id12);
if (dxl_addparam_result != true) {
  ROS_ERROR("Failed to addparam to groupSyncRead for Dynamixel ID %d", req.id12);
  return 0;
}
```

```
dxl_comm_result = groupSyncRead.txRxPacket();
if (dxl_comm_result == COMM_SUCCESS) {
  position1 = groupSyncRead.getData((uint8_t)req.id1, ADDR_PRESENT_POSITION, 2);
  position2 = groupSyncRead.getData((uint8_t)req.id2, ADDR_PRESENT_POSITION, 2);
  position3 = groupSyncRead.getData((uint8_t)req.id3, ADDR_PRESENT_POSITION, 2);
  position4 = groupSyncRead.getData((uint8_t)req.id4, ADDR_PRESENT_POSITION, 2);
  position5 = groupSyncRead.getData((uint8_t)req.id5, ADDR_PRESENT_POSITION, 2);
  position6 = groupSyncRead.getData((uint8_t)req.id6, ADDR_PRESENT_POSITION, 2);
  position7 = groupSyncRead.getData((uint8_t)req.id7, ADDR_PRESENT_POSITION, 2);
  position8 = groupSyncRead.getData((uint8_t)req.id8, ADDR_PRESENT_POSITION, 2);
  position9 = groupSyncRead.getData((uint8_t)req.id9, ADDR_PRESENT_POSITION, 2);
  position10 = groupSyncRead.getData((uint8_t)req.id10, ADDR_PRESENT_POSITION, 2);
  position11 = groupSyncRead.getData((uint8_t)req.id11, ADDR_PRESENT_POSITION, 2);
  position12 = groupSyncRead.getData((uint8_t)req.id12, ADDR_PRESENT_POSITION, 2);


  ROS_INFO("getPosition : [POSITION:%d]", position1);
  ROS_INFO("getPosition : [POSITION:%d]", position2);
  ROS_INFO("getPosition : [POSITION:%d]", position3);
  ROS_INFO("getPosition : [POSITION:%d]", position4);
  ROS_INFO("getPosition : [POSITION:%d]", position5);
  ROS_INFO("getPosition : [POSITION:%d]", position6);
  ROS_INFO("getPosition : [POSITION:%d]", position7);
  ROS_INFO("getPosition : [POSITION:%d]", position8);
  ROS_INFO("getPosition : [POSITION:%d]", position9);
  ROS_INFO("getPosition : [POSITION:%d]", position10);
  ROS_INFO("getPosition : [POSITION:%d]", position11);
  ROS_INFO("getPosition : [POSITION:%d]", position12);
```

```
    res.position1 = position1;
    res.position2 = position2;
    res.position3 = position3;
    res.position4 = position4;
    res.position5 = position5;
    res.position6 = position6;
    res.position7 = position7;
    res.position8 = position8;
    res.position9 = position9;
    res.position10 = position10;
    res.position11 = position11;
    res.position12 = position12;

    groupSyncRead.clearParam();
    return true;
  } else {
    ROS_ERROR("Failed to get position! Result: %d", dxl_comm_result);
    groupSyncRead.clearParam();
    return false;
  }
}

void syncSetPositionCallback(const dynamixel_sdk_examples::SyncSetPosition::ConstPtr & msg)
{
  uint8_t dxl_error = 0;
  int dxl_comm_result = COMM_TX_FAIL;
  int dxl_addparam_result = false;
  uint8_t param_goal_position1[4];
  uint8_t param_goal_position2[4];
  uint8_t param_goal_position3[4];
  uint8_t param_goal_position4[4];
  uint8_t param_goal_position5[4];
  uint8_t param_goal_position6[4];
  uint8_t param_goal_position7[4];
  uint8_t param_goal_position8[4];
  uint8_t param_goal_position9[4];
  uint8_t param_goal_position10[4];
  uint8_t param_goal_position11[4];
  uint8_t param_goal_position12[4];
```

```
  // Position Value of X series is 4 byte data. For AX & MX(1.0) use 2 byte data(uint16_t) for the Position Value.

  //Dynamixel_1
  uint16_t position1 = (unsigned int)msg->position1; // Convert int32 -> uint32
  param_goal_position1[0] = DXL_LOBYTE(DXL_LOWORD(position1));
  param_goal_position1[1] = DXL_HIBYTE(DXL_LOWORD(position1));
  param_goal_position1[2] = DXL_LOBYTE(DXL_HIWORD(position1));
  param_goal_position1[3] = DXL_HIBYTE(DXL_HIWORD(position1));

  //Dynamixel_2
  uint16_t position2 = (unsigned int)msg->position2; // Convert int32 -> uint32
  param_goal_position2[0] = DXL_LOBYTE(DXL_LOWORD(position2));
  param_goal_position2[1] = DXL_HIBYTE(DXL_LOWORD(position2));
  param_goal_position2[2] = DXL_LOBYTE(DXL_HIWORD(position2));
  param_goal_position2[3] = DXL_HIBYTE(DXL_HIWORD(position2));

  //Dynamixel_3
  uint16_t position3 = (unsigned int)msg->position3; // Convert int32 -> uint32
  param_goal_position3[0] = DXL_LOBYTE(DXL_LOWORD(position3));
  param_goal_position3[1] = DXL_HIBYTE(DXL_LOWORD(position3));
  param_goal_position3[2] = DXL_LOBYTE(DXL_HIWORD(position3));
  param_goal_position3[3] = DXL_HIBYTE(DXL_HIWORD(position3));

  //Dynamixel4
  uint16_t position4 = (unsigned int)msg->position4; // Convert int32 -> uint32
  param_goal_position4[0] = DXL_LOBYTE(DXL_LOWORD(position4));
  param_goal_position4[1] = DXL_HIBYTE(DXL_LOWORD(position4));
  param_goal_position4[2] = DXL_LOBYTE(DXL_HIWORD(position4));
  param_goal_position4[3] = DXL_HIBYTE(DXL_HIWORD(position4));

  //Dynamixel_5
  uint16_t position5 = (unsigned int)msg->position5; // Convert int32 -> uint32
  param_goal_position5[0] = DXL_LOBYTE(DXL_LOWORD(position5));
  param_goal_position5[1] = DXL_HIBYTE(DXL_LOWORD(position5));
  param_goal_position5[2] = DXL_LOBYTE(DXL_HIWORD(position5));
  param_goal_position5[3] = DXL_HIBYTE(DXL_HIWORD(position5));
```

```
//Dynamixel_6
uint16_t position6 = (unsigned int)msg->position6; // Convert int32 -> uint32
param_goal_position6[0] = DXL_LOBYTE(DXL_LOWORD(position6));
param_goal_position6[1] = DXL_HIBYTE(DXL_LOWORD(position6));
param_goal_position6[2] = DXL_LOBYTE(DXL_HIWORD(position6));
param_goal_position6[3] = DXL_HIBYTE(DXL_HIWORD(position6));

//Dynamixel_7
uint16_t position7 = (unsigned int)msg->position7; // Convert int32 -> uint32
param_goal_position7[0] = DXL_LOBYTE(DXL_LOWORD(position7));
param_goal_position7[1] = DXL_HIBYTE(DXL_LOWORD(position7));
param_goal_position7[2] = DXL_LOBYTE(DXL_HIWORD(position7));
param_goal_position7[3] = DXL_HIBYTE(DXL_HIWORD(position7));

//Dynamixel_8
uint16_t position8 = (unsigned int)msg->position8; // Convert int32 -> uint32
param_goal_position8[0] = DXL_LOBYTE(DXL_LOWORD(position8));
param_goal_position8[1] = DXL_HIBYTE(DXL_LOWORD(position8));
param_goal_position8[2] = DXL_LOBYTE(DXL_HIWORD(position8));
param_goal_position8[3] = DXL_HIBYTE(DXL_HIWORD(position8));

//Dynamixel_9
uint16_t position9 = (unsigned int)msg->position1; // Convert int32 -> uint32
param_goal_position9[0] = DXL_LOBYTE(DXL_LOWORD(position9));
param_goal_position9[1] = DXL_HIBYTE(DXL_LOWORD(position9));
param_goal_position9[2] = DXL_LOBYTE(DXL_HIWORD(position9));
param_goal_position9[3] = DXL_HIBYTE(DXL_HIWORD(position9));

//Dynamixel_10
uint16_t position10 = (unsigned int)msg->position10; // Convert int32 -> uint32
param_goal_position10[0] = DXL_LOBYTE(DXL_LOWORD(position10));
param_goal_position10[1] = DXL_HIBYTE(DXL_LOWORD(position10));
param_goal_position10[2] = DXL_LOBYTE(DXL_HIWORD(position10));
param_goal_position10[3] = DXL_HIBYTE(DXL_HIWORD(position10));
```

```
//Dynamixel_11
uint16_t position11 = (unsigned int)msg->position11; // Convert int32 -> uint32
param_goal_position11[0] = DXL_LOBYTE(DXL_LOWORD(position11));
param_goal_position11[1] = DXL_HIBYTE(DXL_LOWORD(position11));
param_goal_position11[2] = DXL_LOBYTE(DXL_HIWORD(position11));
param_goal_position11[3] = DXL_HIBYTE(DXL_HIWORD(position11));

//Dynamixel_12
uint16_t position12 = (unsigned int)msg->position12; // Convert int32 -> uint32
param_goal_position12[0] = DXL_LOBYTE(DXL_LOWORD(position12));
param_goal_position12[1] = DXL_HIBYTE(DXL_LOWORD(position12));
param_goal_position12[2] = DXL_LOBYTE(DXL_HIWORD(position12));
param_goal_position12[3] = DXL_HIBYTE(DXL_HIWORD(position12));

// Write Goal Position (length : 4 bytes)
// When writing 2 byte data to AX / MX(1.0), use write2ByteTxRx() instead.

//Dynamixel_1
dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id1, param_goal_position1);
if (dxl_addparam_result != true) {
  ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id1);
}

//Dynamixel_2
dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id2, param_goal_position2);
if (dxl_addparam_result != true) {
  ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id2);
}

//Dynamixel_3
dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id3, param_goal_position3);
if (dxl_addparam_result != true) {
  ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id3);
}
```

```
 //Dynamixel_4
 dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id4, param_goal_position4);
 if (dxl_addparam_result != true) {
   ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id4);
 }

 //Dynamixel_5
 dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id5, param_goal_position5);
 if (dxl_addparam_result != true) {
   ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id5);
 }

 //Dynamixel_6
 dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id6, param_goal_position6);
 if (dxl_addparam_result != true) {
   ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id6);
 }

 //Dynamixel_7
 dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id7, param_goal_position7);
 if (dxl_addparam_result != true) {
   ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id7);
 }

 //Dynamixel_8
 dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id8, param_goal_position8);
 if (dxl_addparam_result != true) {
   ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id8);
 }

 //Dynamixel_9
 dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id9, param_goal_position9);
 if (dxl_addparam_result != true) {
   ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id9);
 }
```

```
//Dynamixel_10
dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id10, param_goal_position10);
if (dxl_addparam_result != true) {
  ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id10);
}

//Dynamixel_11
dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id11, param_goal_position11);
if (dxl_addparam_result != true) {
  ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id11);
}

//Dynamixel_12
dxl_addparam_result = groupSyncWrite.addParam((uint8_t)msg->id12, param_goal_position12);
if (dxl_addparam_result != true) {
  ROS_ERROR( "Failed to addparam to groupSyncWrite for Dynamixel ID %d", msg->id12);
}

dxl_comm_result = groupSyncWrite.txPacket();
if (dxl_comm_result == COMM_SUCCESS) {
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id1, msg->position1);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id2, msg->position2);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id3, msg->position3);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id4, msg->position4);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id5, msg->position5);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id6, msg->position6);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id7, msg->position7);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id8, msg->position8);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id9, msg->position9);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id10, msg->position10);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id11, msg->position11);
  ROS_INFO("setPosition : [ID:%d] [POSITION:%d]", msg->id12, msg->position12);

} else {
  ROS_ERROR("Failed to set position! Result: %d", dxl_comm_result);
}
```

```
  groupSyncWrite.clearParam();
}

int main(int argc, char ** argv)
{
  uint8_t dxl_error = 0;
  int dxl_comm_result = COMM_TX_FAIL;

  if (!portHandler->openPort()) {
    ROS_ERROR("Failed to open the port!");
    return -1;
  }

  if (!portHandler->setBaudRate(BAUDRATE)) {
    ROS_ERROR("Failed to set the baudrate!");
    return -1;
  }

  //Dynamixel_1
  dxl_comm_result = packetHandler->write1ByteTxRx(
    portHandler, DXL1_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS) {
    ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL1_ID);
    return -1;
  }

  //Dynamixel_2
  dxl_comm_result = packetHandler->write1ByteTxRx(
    portHandler, DXL2_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
  if (dxl_comm_result != COMM_SUCCESS) {
    ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL2_ID);
    return -1;
  }
 //Dynamixel_3
 dxl_comm_result = packetHandler->write1ByteTxRx(
   portHandler, DXL3_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
 if (dxl_comm_result != COMM_SUCCESS) {
   ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL3_ID);
   return -1;
 }

 //Dynamixel_4
 dxl_comm_result = packetHandler->write1ByteTxRx(
   portHandler, DXL4_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
 if (dxl_comm_result != COMM_SUCCESS) {
   ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL4_ID);
   return -1;
 }

 //Dynamixel_5
 dxl_comm_result = packetHandler->write1ByteTxRx(
   portHandler, DXL5_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
 if (dxl_comm_result != COMM_SUCCESS) {
   ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL5_ID);
   return -1;
 }

 //Dynamixel_6
 dxl_comm_result = packetHandler->write1ByteTxRx(
   portHandler, DXL6_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
 if (dxl_comm_result != COMM_SUCCESS) {
   ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL6_ID);
   return -1;
 }

 //Dynamixel_7
 dxl_comm_result = packetHandler->write1ByteTxRx(
   portHandler, DXL7_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
 if (dxl_comm_result != COMM_SUCCESS) {
   ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL7_ID);
   return -1;
 }
```

```
//Dynamixel_8
dxl_comm_result = packetHandler->write1ByteTxRx(
  portHandler, DXL8_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS) {
  ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL8_ID);
  return -1;
}

//Dynamixel_9
dxl_comm_result = packetHandler->write1ByteTxRx(
  portHandler, DXL9_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS) {
  ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL9_ID);
  return -1;
}

//Dynamixel_10
dxl_comm_result = packetHandler->write1ByteTxRx(
  portHandler, DXL10_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS) {
  ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL10_ID);
  return -1;
}

//Dynamixel_11
dxl_comm_result = packetHandler->write1ByteTxRx(
  portHandler, DXL11_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS) {
  ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL11_ID);
  return -1;
}

//Dynamixel_12
dxl_comm_result = packetHandler->write1ByteTxRx(
  portHandler, DXL12_ID, ADDR_TORQUE_ENABLE, 1, &dxl_error);
if (dxl_comm_result != COMM_SUCCESS) {
  ROS_ERROR("Failed to enable torque for Dynamixel ID %d", DXL12_ID);
  return -1;
}
```

```
  ros::init(argc, argv, "sync_read_write_node");
  ros::NodeHandle nh;
  ros::ServiceServer sync_get_position_srv = nh.advertiseService("/sync_get_position", syncGetPresentPositionCallback);
  ros::Subscriber sync_set_position_sub = nh.subscribe("/sync_set_position", 10, syncSetPositionCallback);
  ros::spin();

  portHandler->closePort();
  return 0;
}
```