Dissertation

Control System for a Humanoid Robot

A dissertation submitted in partial satisfaction of the
Requirements for the degree of
Doctor of Science
In
Electrical Engineering

Advisor:

Univ. Prof. Dr.techn. Dr. h. c. mult. Peter Kopacek

Faculty of Electrical Engineering and Information Technology
In the
Vienna University of Technology (Austria)

By:

Ahmad Byagowi

Student Number: e0727851

Vienna, March 2010

Abstract

In this thesis, the design and implementation of a control system for a humanoid robot is presented. The control system is implemented for a teen sized humanoid robot, named Archie. Project Archie started at the Institute of Handling Robots and Technology from Vienna University of Technology in the year 2004. The aim of this project is to construct a robot that can imitate human movements such as walking.

The control system is designed based on distributed computer architecture, which means that the entire control system consist of multiple individual motion control units which in turn control the joints (i.e., each joint has a motion controller) and communicate through a data network with the central controller. The central controller is designed on a system-on-chip, based on embedded systems. In this system, an embedded processor and some peripheral hardware result a minimum system to execute a standard operating system (Real time Linux).

The joint controllers of the robot face different load properties based on the overall pose of the robot. Since the design of the motion controllers are based on the load specification, tuning the motion controller parameters is a necessary task. This task can be simplified by using a simulation of the robot. The simulation will anticipate the operation of the motion controllers in the real robot.

Finally, the control system of the real robot is tested and evaluated by the traversed trajectories and they are compared with the simulation of the robot. The trajectories of the real robot are taken using image processing from a video stream of the robot's movement. The evaluation is done by comparing the simulation errors and the error obtained from the data of the real robot on the same test.

Acknowledgments

Hereby I would like to thank my supervisor, Professor Peter Kopacek, for his exceptional patience and understanding with my questions and for giving me the opportunity to explore one of the most interesting fields of robotics (humanoid robots). His experiences helped me to overcome the obstacles that arose in this research, in a constructive and positive way.

In addition, I would like to thank other members of IHRT, namely Dr. Bernhard Putz and Mr. Peter Unterkreuter for their willingness to help me.

I would also like to thank and appreciate all the students of artificial agent lab (AALAB) and Professor Baltes from University of Manitoba (Canada) for their great help and advice during the project. Also, I would like to thank Ms. Ureña Ramírez for helping me during the time of writing my thesis.

Finally, my most profound thanks are to my father, Professor Zaki Byagowi who is the inspiration for my life and my mother, Asma Hosseini who has always given me her infinite love and devotion. I also want to thank my sister, Shima Byagowi and my brother, Ebrahim Byagowi for their love and understanding, support and trust that they all showed me all my life.

Ahmad Byagowi

Table of Contents

Table of Figures

List of Tables

List of Acronyms

ASIC	-Application specific integrated circuit
ASIMO	-Advanced step in innovation mobility
BCS	-Base coordinate system
BLDC	-Brushless direct current motor
CM	-Center of mass
CP	-Center of pressure
CG	-Center of gravity
DAU	-Data acquisition unit
DH	-Denavit Hartenberg
DOF	-Degrees of freedom
DPRAM	-Dual port random access memory
DSP	-Digital signal processor
DSP	-Double support phase
EDK	-Embedded development kit
ENC	-Encoder
FPGA	-Fundamental programmable gate array
GCM	-Ground projection of center of mass
GuRoo	-Grossly underfunded Roo
SPSA	-Sole pressure sensor array
HT	-Homogeneous transformation
IHRT	-Institute of handling robotics and technology
IMU	-Inertial measurement unit
IrDA	-Infrared Data Association
LAN	-Local area network
MOSFET	-Metal oxide semiconductor field effect transistor
OS	-Operating system
PCB	-Printed circuit board
PLB	-Processor local bus
PWM	-Pulse width modulation
RAM	-Random access memory
RAMDAC	-Random access memory digital-to-analog converter
RT-OS	-Real time operating system
SD	-Secured digital
SDR	-Sony dream robot
SPI	-Serial peripheral interface
SSP	-Single support phase
USB	-Universal serial bus
VGA	-Video graphic adapter
VHDL	-Very high integrated hardware description language
ZMP	-Zero moment point

Problem Formulation

This thesis will establish a control system for a humanoid robot. Humanoid robots are sophisticated machines, because of the high degree of freedom. Controlling a humanoid robot requires mechanical, electrical, control and software knowledge. The control system entails mechanical perception of the system (humanoid robot).

The control system which includes the hardware (mostly electronic based hardware) and the software algorithm should provide the robot the ability to imitate human walking. Furthermore the hardware should give appropriate processing resource and performance for the control algorithm used in the robot. In addition, the hardware should be optimized to reduce the power consumption of the system which is important in mobile robots (due to the energy source limitations).

As a result of the high processing load on the control system of the robot, appropriate system architecture should be selected in order to enhance the processing performance. The selected Architecture should provide enough reliability and durability. By using a distributed system with a central controller, the robot could have the flexibility and the ability for further developments and extensions.

Since a humanoid robot is a complex machine, its development is a fairly difficult task. A simulator can be used to ease the development task. Moreover, using a simulator could reduce the development time as well as cost (e.g., reducing system impairments caused during development failures).

To evaluate the robot's performance in real world; for instance, the movement trajectories a solution should be taken such that it provides enough accuracy and reliability. Image processing could be the simple and reliable method as an approach for solving this problem.

By adding sole pressure sensors to the robot's feet, it should have the possibility to imitate natural human walk, and the ability to walk on non-even terrains in future works.

Chapter 1

1. Introduction

"We can't solve problems by using the same kind of thinking we used when we created them."

- Albert Einstein

Many aspects of modern life involve the use of intelligent machines capable of operating under dynamic interaction with their environment. In view of this, the field of biped locomotion is of special interest when human-like robot is concerned. Currently, research on humanoid robots and biped locomotion is one of the most exciting topics in the field of robotics. The field of robotics is one of the most innovative in the last decade (D. Katic and M. Vukobratovic, 2004).

A humanoid robot is a robot with its overall appearance based on that of the human body, allowing interaction with made-for-human tools or environments. In general, humanoid robots have a torso with a head, two arms and two legs, although some types of humanoid robots may model only part of the body, for instance, from the waist up. Some humanoid robots may also have a face, with eyes and mouth. Androids are humanoid robots built to aesthetically resemble a human.

Nowadays, humanoid robots are often shown in science fiction movies (e.g. I-robot, Ironman and AI), although those robots are far from being able to use in reality. There are a wide range of applications which can be carried out by humanoid robots and not only the ones that are commonly presented by the motion picture industry. For example, humanoid robots can also be used in detecting land mines (A. Byagowi & P. Kopacek, 2009).

In this work, implementation of a relatively low cost control system for a humanoid robot with lower energy consumption is presented. Moreover, this project introduces novel methods which are a combination of mechanics and electronics. The robot's control system is based on distributed computer architecture. A distributed computer consists of multiple computers (controller units) that communicate through a data network with a central controller. The computers interact with each other in order to achieve a common goal (Andrews, Gregory R. 2000).

Using a distributed computer system gives the robot the ability to have real time reactions. In this approach, the control system is divided into levels to break down the heavy processing load besides of increasing the flexibility and reliability of the system. In a distributed computer a problem is divided into many tasks, each of which is solved by one computer (Godfrey C., 2002).

Each joint controller of the robot entails an individual micro controller (Infineon, XC164, 2001). The controller will monitor the motion of the joint (e.g. torque, velocity and the position). Using the data network of the robot, all the individual controllers communicate with the central controller which in turn collects all the data from the joints and makes the calculations that are used to balance the robot. Then, it sends back the appropriate commands to the joint controllers. The central controller (spinal board) is similar to the brain's cerebellum (parts of the vertebrate's brain that controls the balance).

Humanoid walking algorithms can be distinguished as being either static or dynamic. The distinction is made depending on the location of the centre of mass during motion. For static walking, the centre of mass is always located above a polygon created by external boundaries of the leg base. The robot will remain statically stable if it is paused at any time during its motion.

Dynamic walking is generally much faster than static walking. In dynamic walking, inertia effects are considered, and it is possible for the Centre of Mass (CM) to be outside the supporting area. Human walking patterns are considered to be dynamic. Traditionally, robots have maintained stability throughout their motion by maintaining at least three points of contact with the ground at all time. Since humanoid robots have only up to two points of contact with the ground, they must maintain stability through alternative means.

The center of mass for a rigid body is a fixed point which is located near or inside the object. When two objects are connected to each other using a hinge, the total CM will be the resultant of the location, the mass and the angle between (angle of the hinge) the CM of those two objects. The following criterion can be extended to multiple objects and joints. A humanoid robot can be aimed as multiple rigid bodies (links) connected to each other using some hinges (joints).

To find the total center of mass in a humanoid, first the center of mass for each link of the robot has to be relocated in a Base Coordinate System (BSC). The direct kinematics model of the robot is used for this purpose. This model yields a series of transformation matrices which convert relative positions from one coordinate system to another coordinate system and finally to the BCS. To formulate the following conversion chain, the Denavit–Hartenberg (DH) notation is used. This one provides the mathematical calculations for the relocation positions from one coordinate system to the other from multiple conversion matrixes.

A commonly used convention to select the frames of reference in robotics applications is the Denavit-Hartenberg convention (Jaques Denavit, Richard S. Hartenberg, 1989). In this convention, each homogeneous transformation is represented as a product of four basic

transformations. The common normal between two lines was the main geometric concept that allowed Denavit-Hartenberg to find a minimal representation.

The ground projection of the CM (which is resembled using center of pressure) should be above a supporting polygon in order to save the balance for the robot in static walk. The support polygon changes when the points in touch with the ground are modified. During walking, the points in touch of the robot are changing based on the walking phases. Walking consists of two major phases, Single Support Phase (SSP) and Double Support Phase (DSP).

The central controller generates the movements using interpolation and controls the balance of the robot. To maintain the balance in the robot, the center of mass has to be calculated. Furthermore, the joints have to be moved in a suitable way to shift the ground projection of the CM in the support polygon. This one is in turn related to the phase that the robot is in at some specific time.

In a mobile robot like a humanoid, using optimized hardware that has minimal energy consumption is a critical requirement. The main reason to optimize the hardware's energy consumption is that the battery is a limited energy source. One of the other issues that should be in mind in the design of a humanoid robot is the physical space limit for the control hardware and other possible accessories.

Communication with all the entire individual controllers is a time consuming task that needs processing resources. Using a custom hardware design improves the performance of the system. In this work, a customized hardware, named Data Acquisition Unit (DAU) is presented. The DAU sends data to the individual controllers on the data network. After that, the data from the individual controllers will be captured and collected by the DAU in order to be used by the central controller. The entire process will be executed by the first controller consequentially (i.e., one by one until the last one). All the sent and received data will be exchanged using a Dual-Port RAM (DPRAM) with the main controller (main controller bus), to prevent any interference with the main system (the central controller's processor).

The DAU is implemented on a Fundamental Programmable Gate Array (FPGA) using Very high integrated Hardware Description Language (VHDL). The FPGA contains a hardcore embedded processor (Xilinx Virtex 4, PowerPC 405) which is also used to run the operating system (Embedded real time Linux) to control the entire robot.

1.1. Purpose of this thesis

The purpose of this project is to design and implement a control system for Archie, a tall humanoid robot. The control system which is presented in this project is designed based on embedded systems. Using an embedded system to control a humanoid robot can be beneficial from different aspects. Reducing energy consumption, decreasing the hardware price and increasing the reliability of the control system are some of the benefits of using an embedded system. In this work the algorithms used to balance a humanoid robot during walking are described, then a derivation of the algorithms are provided for the robot that is used in this project (Archie). In addition to the robot, as simulator is performed such that predicts the movement of the real robot in a simulation environment. The simulation can be either reduced or extended based on certain tests.

Finally, some practical tests are performed on the real robot as well as its simulation. The results obtained from the test on the real robot are compared with the simulation of the robot in order to perform a system evaluation.

1.2. Chapter outline

This thesis is divided into seven chapters. A brief description of the contents of each chapter is as follows:

Chapter 1 – Introduction of the work and the control algorithms used in this robot.

Chapter 2 – Introduces relevant information within the field of study. The previous works on other humanoids around the world are introduced.

Chapter 3 –Brief introduction about the humanoid robot Archie and some specifications of the robot; the height, the weight, minimum operation time that the robot can have per each full charging, walking speed and the degree of freedom.

Chapter 4 – Presents the control algorithms used in robot in order to give the ability to walk by saving the balance of the robot. It also talks about the control system used in the joints of the robot and the motion generator algorithm for imitating human like walking.

Chapter 5 – Describes the hardware implementation of the control system designed for the robot. Also it described the communication protocol used in the data bus of the robot beside the physical layer specification. At the end the simulator used for the robot is introduced.

Chapter 6 – Shows the test results for the control system of the robot in three levels; first one joint controller is tested individually, then the joint controllers are combined for a single leg and are tested as the second level and finally in the third level the whole robot is tested for walking.

Chapter 7 – Discusses the conclusion of the work and some ideas for the future developments on the robot. The suggestions are presented for each part of the robot separately.

Appendix A – Matlab codes (Direct Kinematics, Inverse Kinematics and Trajectory planner), Image processing code for patch detection and the Embedded Development Kit (EDK) code for the spinal board.

Appendix B – Schematics of the DC motor driver, Brushless motor driver and the spinal board circuits.

Chapter 2

2.State of the art and literature review

This chapter introduces a background on the development of humanoid robots and the methods used in their control systems. To begin with, some famous humanoid robots will be introduced, and then the control systems used in this type of robots will be briefly described.

2.1. Selected examples of humanoid robots

Humanoid robot development is a relatively new field in robotics research and few results are publicly recognized. The most publicized humanoid robots are Honda's ASIMO, Sony's QRIO "SDR-6X", GuRoo, Nao, and Android.

Honda's ASIMO robot (figure 2-1) is 120cm in height and was originally conceived to function in an actual living environment. It has the ability to walk continuously and smoothly while changing direction, and can travel up to 0.44m/s. By predicting its next movement in real time, ASIMO shifts its centre of gravity in anticipation of its path. For balance control, ASIMO uses gyroscopic and accelerative sensors in the torso, as well as 6-axis foot area sensors.

Figure 2-1: Latest generation of ASIMO (Honda, 2005)

Sony's QRIO "SDR-6X" (figure 2-2) is a small biped robot that measures 50 cm in height. It is able to walk at a velocity of 0.33m/s as well as to demonstrate basic movements such as walking and changing direction, standing up, balancing on one leg, kicking a ball and dancing. Their movements allow it to walk on non-even surfaces and in the presence of external forces. To get feedback regarding posture and position control, it uses acceleration sensors in the torso and four pressure sensors on each foot.

Figure 2-2: Sony's QIRO "SDR-6X" (2003)

GuRoo (figure 2-3a) is 120cm in height and it is completely autonomous. It is used for research in different areas including dynamic stability, human-robot interaction and machine learning.

Nao (shown in figure 2-3b) is an autonomous, programmable and medium-sized humanoid robot, created for companionship. Nao RoboCup Edition has 21 Degree of Freedom (DOF) while Nao Academics Edition has 25 DOF since is built with two hands with gripping abilities. Nao features a powerful multimedia system (four microphones, two amplified speakers, two cameras) for text-to-speech synthesis, sound localization or facial and shape recognition amongst various other abilities.

GuRoo and Nao both compete in the annual RoboCup contest. The goal of this competition is to foster the development of robotics through an annual soccer competition. The main goal of the RoboCup federation is to develop a team composed only by autonomous humanoid robots to play against and beat the human team that wins the World Cup in the year 2050.

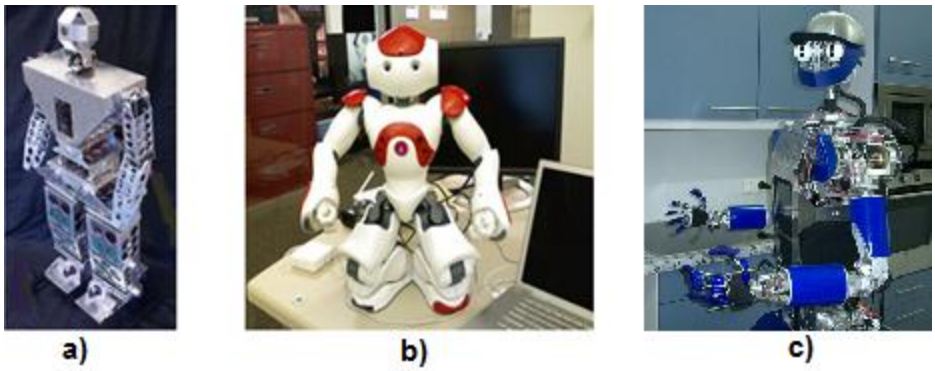


Figure 2-3: a) GuRoo robot, b) Nao robot, c) ARMAR III

An Android is a robot or synthetic organism designed to look and act like a human. Androids have been mainly an element of science fiction, yet they have increasingly become a reality in Japan and South Korea. The two countries are in a heated competition to make them commercial success in the global market and have developed a handful of successful androids so far. In figure 2-4 some examples of android robots are depicted.

Figure 2-4: From left to right: Android, DER 01 and DATA from Star Trek

2.2. Control systems used in humanoid robots

Most of the humanoid robots are based on hierarchical distributed control systems. In this type of controlling, the system (robot) is organized and divided into local parts controlled using

individual controllers. All of the individual controllers are communicated with the main controller to unify their functionality in order to reach a certain purpose for the whole robot.

All the control systems in the humanoid robots are designed with the purpose of saving the balance of the robot and giving the robot the ability to walk and to be standing stable on two legs (sometimes only on one leg). The control algorithms used in this type of robots have some varieties, although they are based on the same principles. In all the control algorithms used in different humanoid robots, a direct kinematics model is used to find the total center of mass and moment of inertia of the robot. Furthermore, the inverse kinematics model is used to apply the changes on the robot. In chapter 4, these models are described with more detail.

For instance, in the robot ARMAR III (T. Asfour et al., 2008) the control architecture consists of three levels: the micro-controller level, the PC level and the PC-network level. The micro-controllers are used to control the motors and establish the communication with a standard PC as a central controller. The central controller uses RT-Linux as operating system. The communication between the micro-controllers and the central controller uses a standard CAN-Bus. The motors (joints) are controlled by PID controller, where it is done independently for each joint.

In the robot ARMAR III, a fuzzy-like module is implemented which sets the parameters of a classical position joint controller depending on the configuration of the overall pose of the robot. Figure 2-5 shows the block diagram of the control system for the ARMAR III.

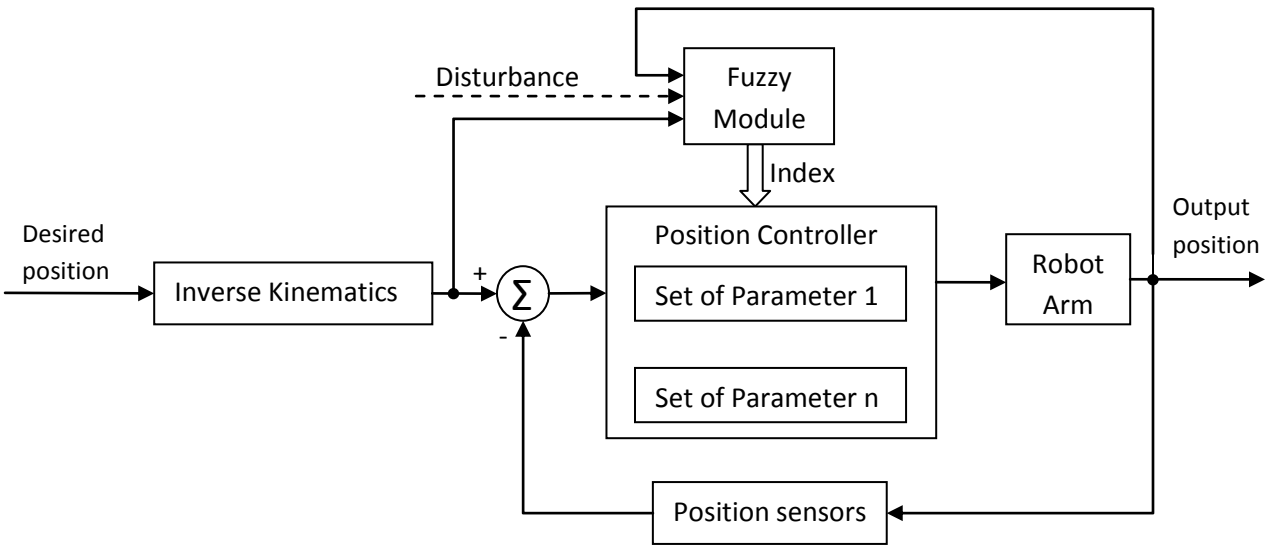


Figure 2-5: ARMAR III‘s control system

Some of the biped robots use Reinforcement learning to control the joints of the robot (Shouyi Wang et al., 2006). This approach relies on the concept of dynamic walking. Dynamic walk is simpler than the static walk. Moreover, dynamic walk can achieve a more natural gait and less energy consumption. Figure 2-6 shows the block diagram for the control system based on reinforcement learning.

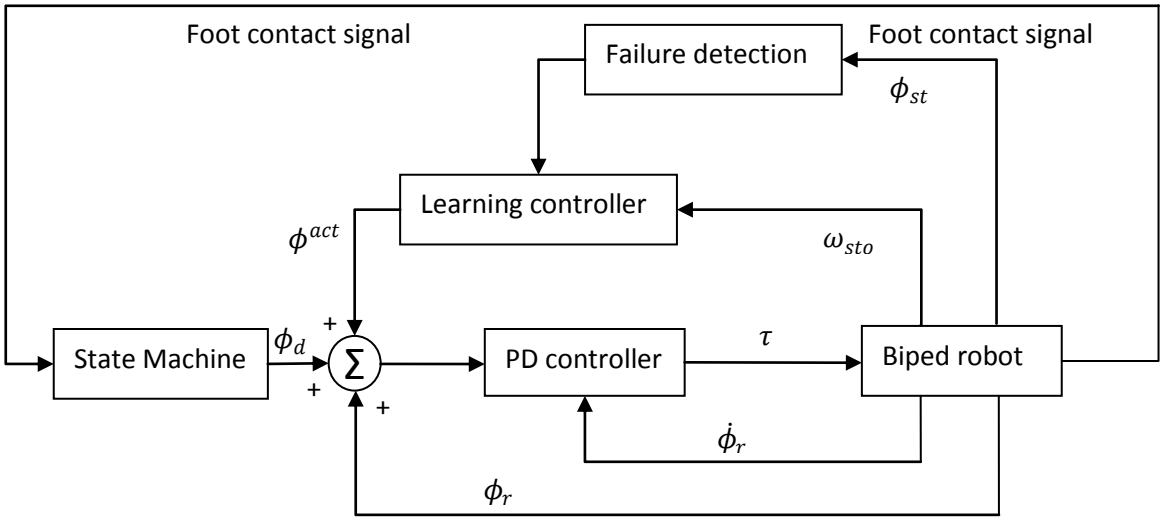


Figure 2-6: The learning control scheme of the biped robot

More advanced control systems for humanoid robots attempt to reproduce and execute the same posture control operation carried out by real humans (T. Takenaka, 2006). One of these robots is the ASIMO from Honda.

The robot ASIMO uses macro stabilization control to avoid tipping over either during walking or in standing position. The macro stabilization control system tries to realize the same posture control operation undertaken by a human being. The resultant force of gravity and inertia force are called the total inertia force. The point where the line of action of the total inertia force intersects with the ground surface is called the Zero Moment Point (ZMP). The point where the ground reaction force acts is known as the Center of Pressure (CP).

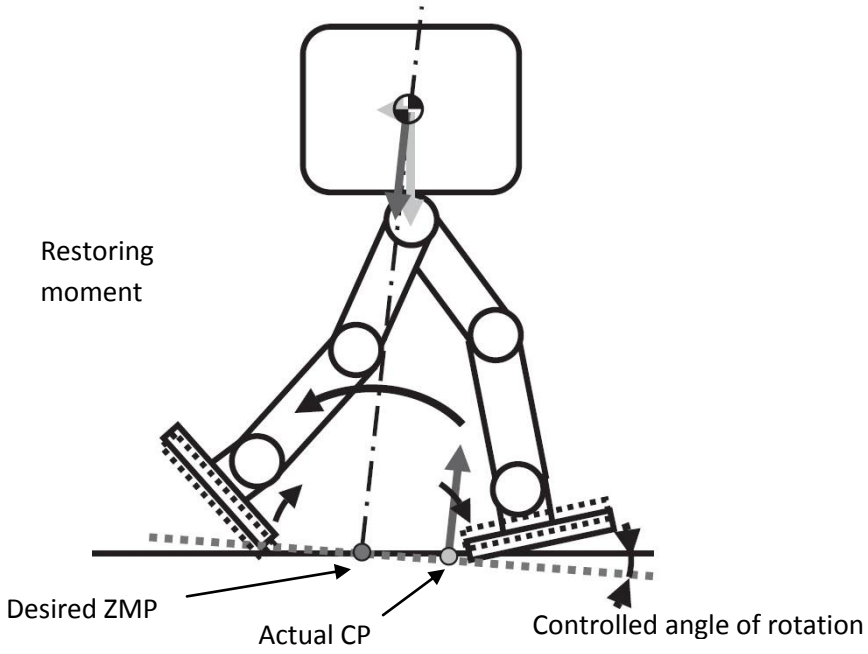


Figure 2-7: Feet rotation around ZMP

The Model ZMP Control uses shifting the desired ZMP to an appropriate position to prevent the robot from tipping over. For example, in case that the robot is in danger of falling forward the model ZMP control accelerates the robot's upper body trajectory more strongly to forward than the supposed acceleration. This reaction results the ZMP to shift backward from the original desired ZMP to an appropriate point behind the actual CP and recovers the moment of the robot (i.e., the Model ZMP Control restores the posture of the robot by intentionally unbalancing the desired walking pattern) (T. Takenaka, 2006).

Chapter 3

3. The humanoid robot Archie

At the Institute of Handling Robotics and Technology (IHRT) of Vienna University of Technology a humanoid robot project is running since 2004. The main purpose is developing and implementing a human like robot (platform). The platform is thought to be of human size, low cost, modular and to perform a natural looking straight gait with the following features:

Height:1500 mm

Weight: less than 20kg

Operating time: minimum 60 minutes

Walking speed: minimum 0.5 m/s

Degrees of freedom: 29

On board intelligence

Dynamic Walking (ZMP)

Hierarchical, decentralized control structure

Reasonable low price – try to use commercially available standard components.

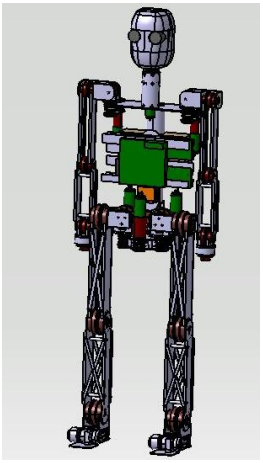


Figure 3-1: Archie's mechanical simulator overview

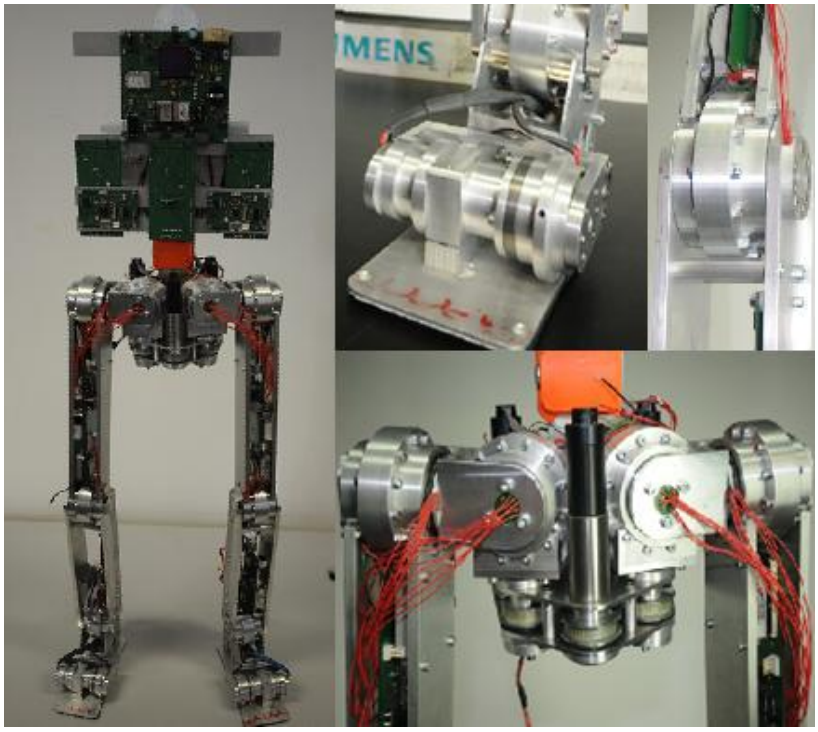


Figure 3-2: Design of Archie

Figure 3-3: Foot design and simulation for the prototyping of the foot

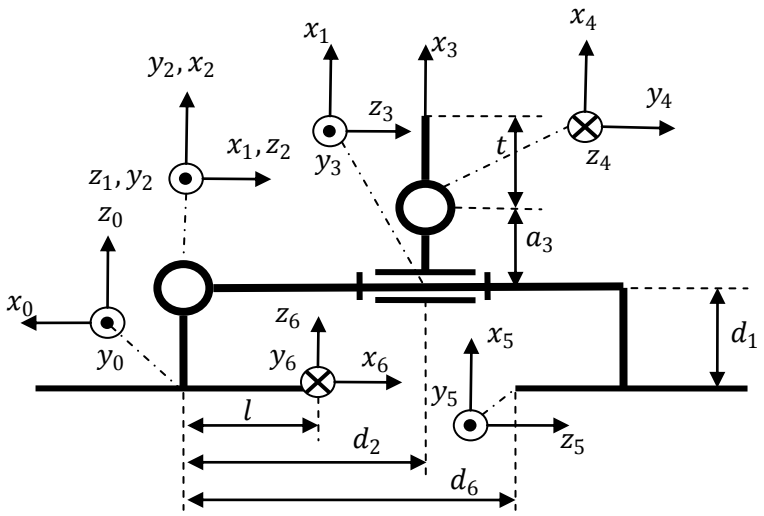


Figure 3-4: Denavit-Hartenberg model for the prototyping of the foot

Archie has 29 servo motors in its body that move its torso, arms, hands, legs, feet, ankles and other moving parts. Archie manages a series of servo motors to control each kind of movement.

Archie is powered by a rechargeable, 29.4 volt lithium ion (Li-ION) battery that lasts for about one and half hour on a single charge. The battery is stored in Archie's flank (left and right) and weighs about three kilograms. Archie's battery takes around three hours to fully charge. Users can charge the battery onboard Archie through a power. During battery charging the robot cannot be used.

The simulator used for the robot is based on the SimMechanics toolbox of Matlab-Simulink. Figure 3-5 illustrates a screen shot of the simulator of the lower body of the robot.

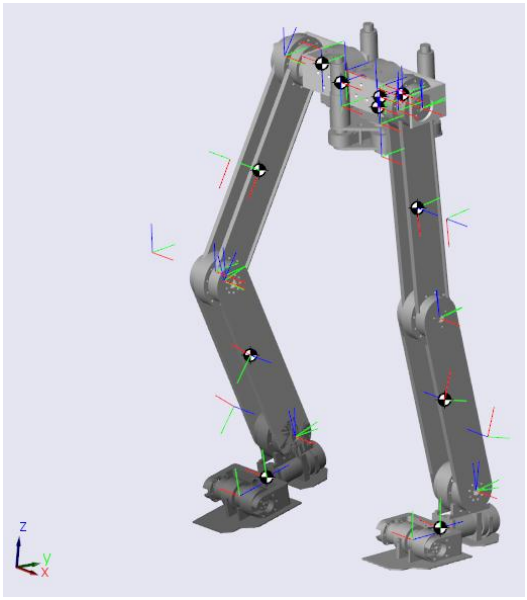


Figure 3-5: Archie's simulator (simulation of the Archie's lower body)

Chapter 4

4. The control concept of Archie

4.1. Introduction

This chapter presents a description of the control system of Archie. The control system in Archie is designed based on the purposes discussed in this chapter. Since in all robotic control systems, mechanical analysis is an important topic, a part of this chapter is about the mechanical analysis of Archie. Archie's control system is based on the distributed architecture. The control system consists of two levels, joint controllers in low level and a central controller in high level. The joint controllers are used to control the motion of each joint and are connected with the central controller through a data network. The central controller synchronizes all the joint controllers, calculates the center of mass and tries to locate it in a position above the supporting polygon to keep the balance of the robot.
In the torso of the robot, an Inertial Measurement Unit (IMU) is mounted to provide a better control performance. Other instruments such as Sole Pressure Sensor Array (SPSA) may also be used in the feet of the robot to provide sufficient control and ability to walk in non-even terrains.

4.2. Control system architecture

The control system used in this robot is based on the distributed architecture. In this structure, each joint is controlled individually by a motion controller which in turn communicates with the central controller via a data network. The central controller is responsible for doing the following tasks:

- Energy management
- Multitask management
- System failure detection

- Performing received commands
- Synchronizing the joint controllers
- Saving the overall balance of the robot
- Calculating the location of the supporting polygon
- Preventing mechanical collision in manual movements
- Updating the desired positions resulted from calculation with the joints
- Splitting general commands into joint commands (for combinational movements)

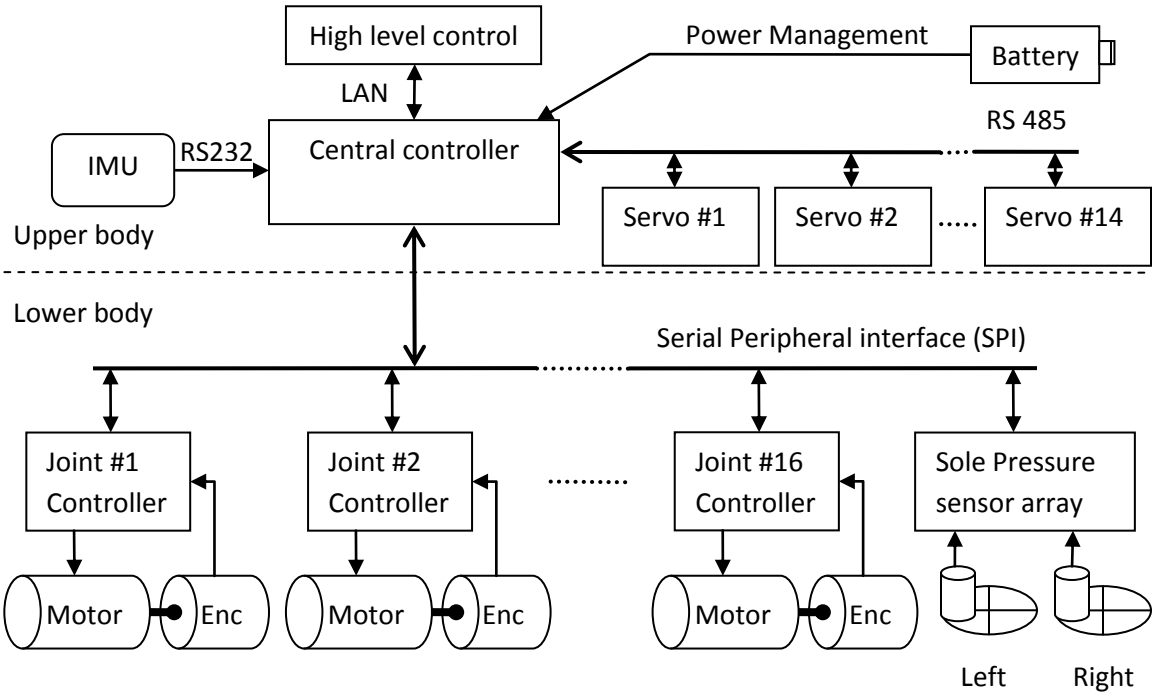


Figure 4-1: Control system block diagram

In figure 4-1 the block diagram of the control system is illustrated. Due to the system demands, the control system should provide appropriate control capabilities. First of all, the control system should have comprehensive information about the mechanical structure such as mass, length and moment of inertia tensor for each link of the robot. The control system uses that information to make a direct kinematics model (figure 4-2) of the robot. The model is then used by the central controller to calculate the total center of mass before issuing appropriate commands and performing movements.

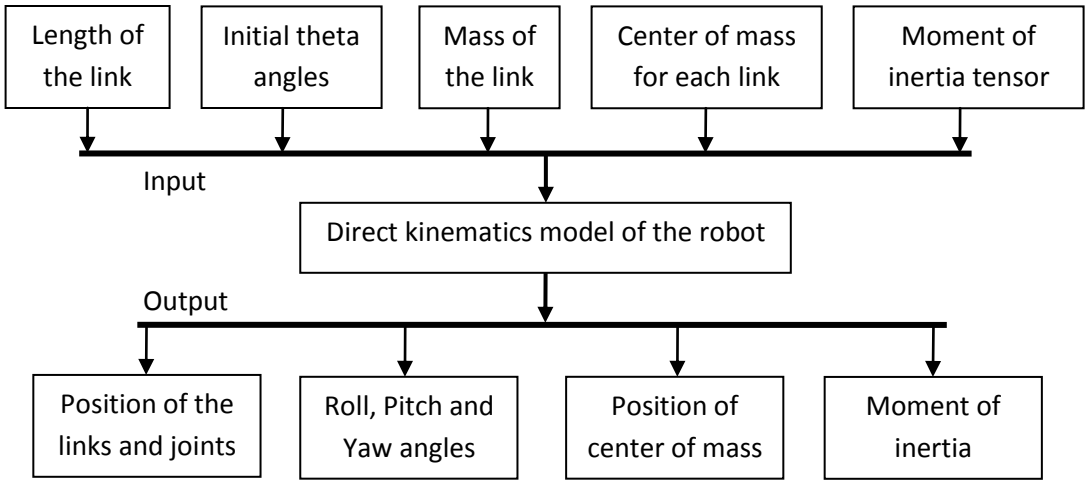


Figure 4-2: Direct kinematics input and output data

The direct kinematics provides some parameters such as moment of inertia, position of the joints and links, Euler angles (roll, pitch and yaw) and the position of center of mass. In order to control the humanoid robot, these parameters should be controlled. Base on the control method of the parameters (i.e., simple control loops or complex control method) the output should be applied appropriately to move the parameter to the desired value. The output of the control loop can be applied using several methods one of which is the inverse kinematics. Figure 4-3 shows a block diagram for the inverse kinematics and the data exchange.

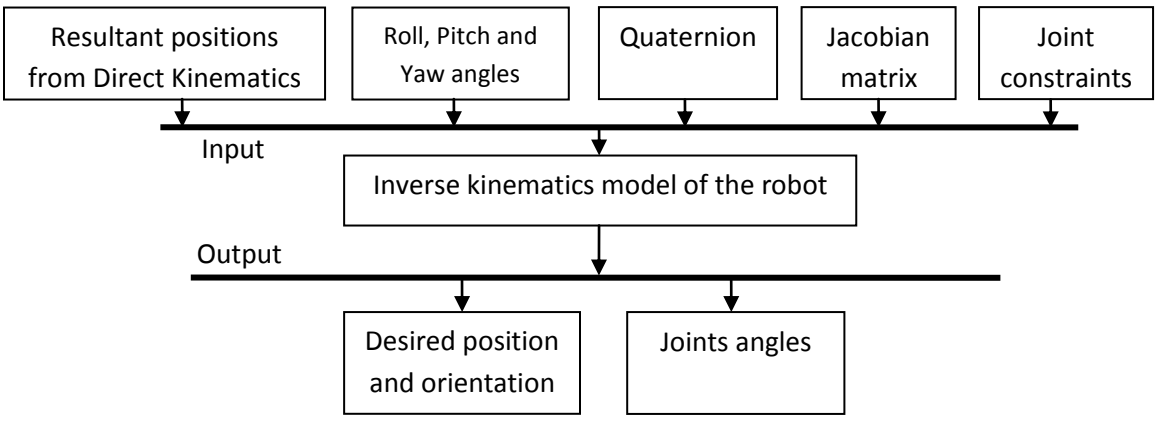


Figure 4-3: Inverse kinematics input and output data

The humanoid robot is a rigid multi-body system consists of a set of rigid object, called joints and links.

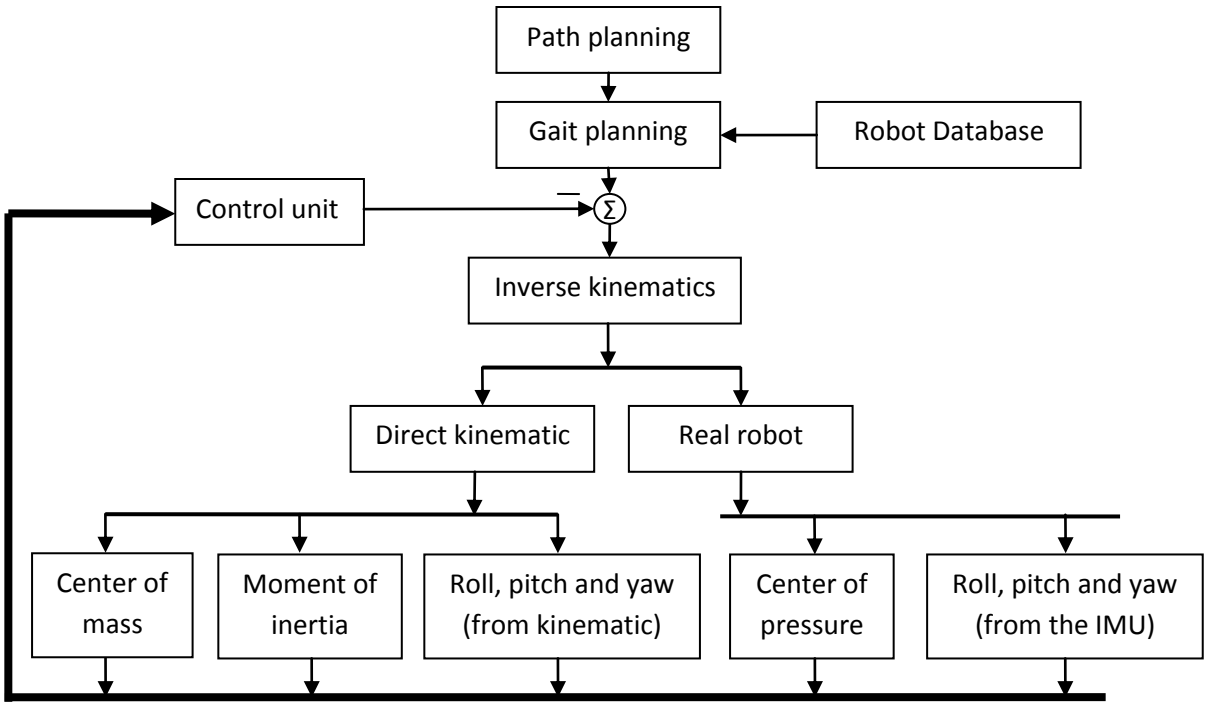


Figure 4-4: Humanoid control system structure

4.3. Mechanical analysis of the humanoid robot

To resemble the links of the robot in the space, the representation of position and orientation is necessary. Resembling a revolute joint with a single Degree of Freedom (DOF) can be done by using a single real number that is the angle of the rotation on that link, relative to an arbitrary zero point (Homing point).

Mechanical analysis for the humanoid robot involves finding the direct kinematics and finding the total center of mass as well as the moment of inertia for the joints. To find the total center of mass in the robot, first the center of mass of each joint is considered. Then the angle of rotation in the joints and the mass of each link is used to calculate the total center of mass. The center of inertia for each joint in the robot is calculated based on the mathematical methods that are presented in this chapter. Following methods and calculations are derived for the humanoid robot and are used in the control system in the robot.

4.3.1. Homogeneous transformation

Regarding a 3D coordinate system {A}, any link ***L*** can be located by a [*3 x 1*] matrix named position vector. To find the orientation of a link, a coordinate system {B} is attached to the link in a known way. The orientation of {B} relative to the reference frame {A} is a linear transformation which is called the rotation matrix or direct cosine matrix (DCM). The rotation matrix transforming from {B} coordinates to {A} coordinates is written as R_B^A (J.J Craig, 2003):

$$R_B^A : A \leftarrow B \qquad \text{Equation 4-1}$$

$$L^A = R_B^A . P^B \qquad \text{Equation 4-2}$$

P^B is the link vector ***L*** seen from frame {B} and P^A is the link vector ***L*** seen from frame {A}.

If the three principal axes of {B} are described by a set of orthogonal unit vectors $\vec{\boldsymbol{\imath}}$, $\vec{\jmath}$ and $\vec{\boldsymbol{k}}$, then the rotation matrix R_B^A can be written as:

$$R_B^A = [\vec{\imath}^A \ \ \vec{\jmath}^A \ \ \vec{k}^A] \qquad \text{Equation 4-3}$$

By using the rotation matrix presented in equation 4-1, the origins of the frame {A} and {B} coincide as illustrated in figure 4-5a, where the frames {A} and {B} are attached to the same link. To attach the frame {B} to the other link (i.e. {c}), two pieces of information are required to define its relative coordinates to the frame {A}; the rotation matrix R_B^A and the vector $L_{B(org)}^A$ pointing to the origin of {b} from frame {A} (J.J Craig, 2003):

$$L^A = R_B^A . L^B + L_{B(org)}^A \qquad \text{Equation 4-4}$$

Figure 4-5b shows, how this transformation can be applied to several frames.

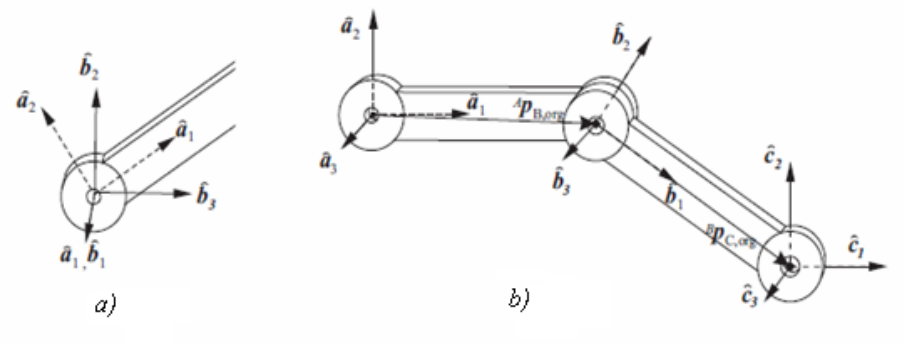


Figure 4-5: a) The origin of {A} and {B} coincides as they are defined relative to each other by a rotation matrix. B) Both a rotation matrix and an offset vector are required to define {B}.

In figure 4-5 the frame {A} is defined relative to {B}, while frame {B} is defined relative to {C}.

For the sake of simplicity, Equation (4.3) is written as a single matrix operation.

$$\begin{bmatrix} \vdots \\ L^A \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} \cdots & \vdots & \cdots & \vdots \\ \cdots & R_B^A & \cdots & L_{B(org)}^A \\ \cdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Equation 4-5}$$

The 4 x 4 matrix in Equation (4.4) is called a homogeneous transform. This transformation matrix T includes all the necessary information about the position and orientation of the reference frame with respect to another frame. An abbreviation of equation 4-4 can be written as it is shown in equation 4-6.

$$A_p = T_B^A . B_p \qquad \text{Equation 4-6}$$

Without indicating *p* is a [4 x 1] vector.

4.3.2. The Denavit-Hartenberg convention

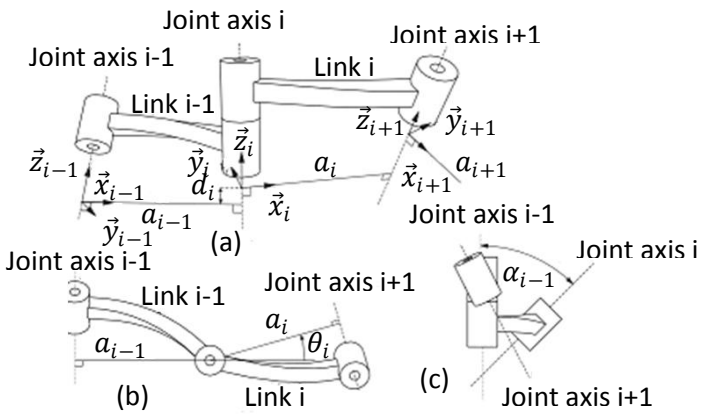
The Denavit-Hartenberg notation is for describing a chain of limbs in a robot. This notation states that the kinematics of a robot can be represented by four parameters for each link: two to describe the link itself and two to describe its connections to the next link. A link is defined as a rigid body which defines the relationship of joint axes of the robot. A joint axis {*i}* is a vector direction in space, which rotates relative to the link {*i-1}*. The relative location of two joint axes can be specified by two parameters; the distance and the angle (or twist) between them. A convention is given for affixing frames to the links of the robot or manipulator which consists of a single chain of links attached to some fixed base:

The third axis z_i of the frame {i} is coincident with the joint axis {i}.

The origin of the frame {i} is located in the place where link length a_i intersects the joint axis {i}.

The first axis x_i points the length of the link a_i in the direction from the joint {i} to {i+1}.

The second axis y_i is formed by the right-hand rule to complete a right-handed coordinate system.



a) The frames are attached using the convention defined in above.

b) The link twist parameter is illustrated for link {*i-1*}.

c) The joint angle for joint {*i*} is illustrated.

Figure 4-6: Link length and link displacement parameters (Xu WL, 1990)

The first link of the chain is the base of the robot, called link {0}. The attached frame is called frame {0} and is stationary. Thus, all other link's positions may be described in terms of this frame. The coordinate system is attached is such way that coincides with frame {1} when the joint angle θ_1 is zero. As a result, the link parameters d_0, a_0 and α_0 are always zero. Using the described frame attachment procedure, the link parameters and joint variable can be found as:

a_i : Distance from z_i to z_{i+1} measured along x_i.

α_i : Angle between z_i to z_{i+1} measured about x_i.

d_i : Distance from x_{i-1} 1 to x_i measured along z_i.

θ_i : Angle between x_i to x_{i+1} about z_i.

The transformation matrix from frame {i} to {i - 1} is found by using from the four link parameters a_i, α_i, d_i and θ_i as it is illustrated in equation 4-7.

$$T_i^{i-1} = \begin{bmatrix} cos\theta_i & -sin\theta_i & 0 & a_{i-1} \\ sin\theta_i . cos\alpha_{i-1} & cos\theta_i . cos\alpha_{i-1} & -sin\alpha_{i-1} & -sin\alpha_{i-1} . d_i \\ sin\theta_i . sin\alpha_{i-1} & cos\theta_i . sin\alpha_{i-1} & cos\alpha_{i-1} & cos\alpha_{i-1} . d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 4-7: Transformation matrix based on Denavit-Hartenberg convention

The four mentioned parameters are generally called: link length, link twist, link offset, and joint angle, respectively. These names are derived from specific aspects of the geometric relationship between the two coordinate frames. Since the matrix A_i is a function of a single variable, three of the above-mentioned four parameters are constant for a given link, while the fourth one (i.e., θ_i, for a revolute joint and d_i for a prismatic joint) is variable.

4.3.3. Mechanical design of Archie

In figure 4-4 the mechanical design of Archie based on the Denavit-Hartenberg vector assigning method is presented. The figure is depicting the lower body of the robot (two legs and the hip). The based coordinate system which is used from unifying the other coordinate systems is located on the hip plane of the robot. The main reason for choosing the hip plane as the base coordinate system is its location because it is in the centre of the robot. Moreover, the inertial measurement unit (IMU) is connected to hip of the robot which can provide the information about the angle of the roll and pitch of the robot to the control system.

The distribution of the mass for the links is considered to be solely concentrated in the centre point of the mass. With this approximation, a link is defined as a rigid connection between coordinate frames and a point mass located somewhere between these frames. The reference frame for each link is located at the centre of the proximal joint.

The transformation matrix from any frame {*i-1*} to {*i*} can be found by inserting the Denavit-Hartenberg parameters (form table 4-1) into the matrix which is shown (equation 4.7). For example, the transformation matrix from the frame {5} to the frame {6} is:

$$T_6^5 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \text{Equation 4-8}$$

Given these matrixes it is possible to compute the position and orientation of all frames relative to each other.

$$T_i^j = \prod_{n=i}^{j} T_{n-1}^n \qquad \text{Equation 4-9}$$

For instance, the transformation matrix is used to calculate the position of a point in the arm frame {*5*} in the base frame {*1*} can be found as:

$$T_5^1 = T_2^1 \times T_3^2 \times T_4^3 \times T_5^4 \qquad \text{Equation 4-10}$$

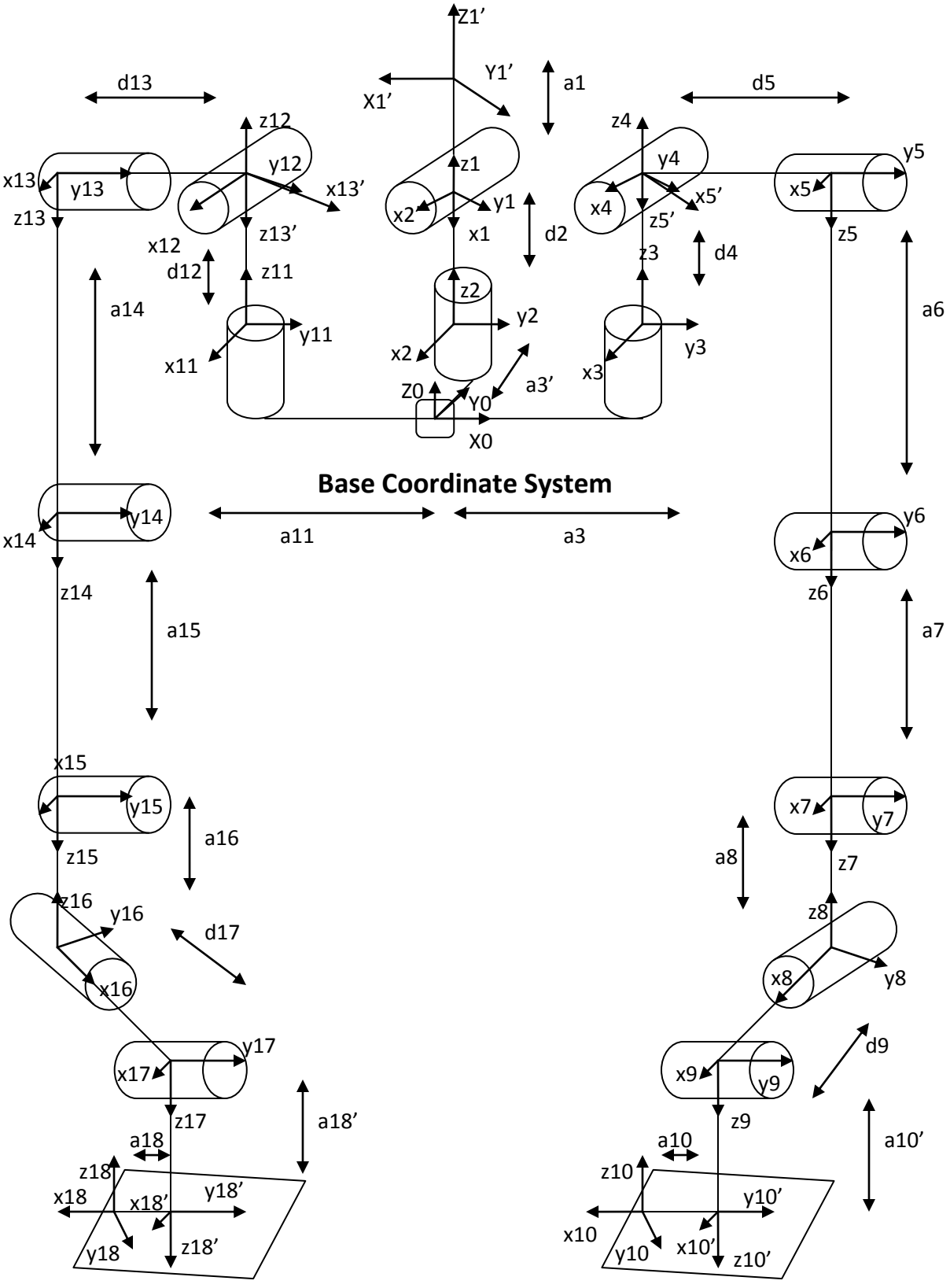


Figure 4-7: Description of the mechanical design of Archie (only lower body)

4.3.4. Deriving the kinematics of Archie

Deriving the kinematics of Archie is carried out in three steps. The first step is to affix frames to the links of the robot. The second step is to identify the four link parameters for each link, and the third is finding the transformation matrixes. The following resulted parameters are called Denavit-Hartenberg (M. Spong, M. Vidyasagar, 1989)

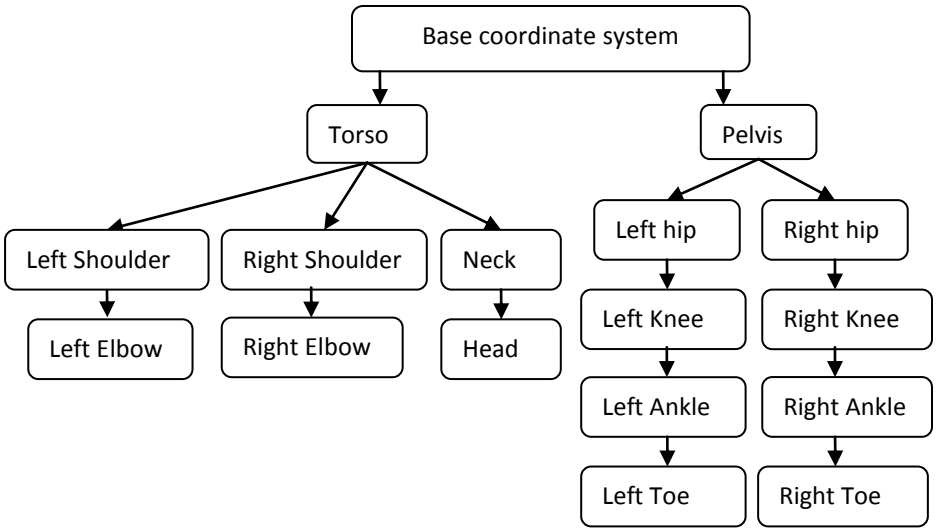


Figure 4-8: Hirarchical graph or Archie's joints

Denavit-Hartenberg (D-H) parameters for Archie are shown in table 4-1. These parameters are captured from the mechanical design of the robot.

Joint	a (cm)	d (cm)	α (degree)	Θ (degree)
1	36	0	-90	-90
2'	0	0	90	90
2	0	5	-180	0
3'	11	0	0	-90
3	4.5	0	0	90
4	0	5	90	0
5'	0	0	-90	-90
5	0	7	0	0
6	31	0	0	0
7	26	0	0	0
8	5.5	0	90	0
9	0	8.4	-90	0
10'	3	0	0	0
10	4	0	180	-90
11	-4.5	0	0	90
12	0	5	90	0
13'	0	0	-90	-90
13	0	-7	0	0
14	31	0	0	0
15	26	0	0	0
16	5.5	0	90	0
17	0	8.4	-90	0
18'	3	0	0	0
18	4	0	180	-90

Table 4-1: Denavit-Hartenberg parameters for Archie

4.3.5. Calculation of the total center of mass

The coordinates of the center of mass for the individual links of the robot are known in their respective joint frames. Thus, the total center of mass of the robot can be found by finding the position of the center of mass for each individual link in the base coordinate system frame can be applied using the following formula:

$$CM_{total} = \frac{1}{m_{total}} \sum m_i CM_i \qquad \text{Equation 4-11}$$

$$m_{total} = \sum m_i \qquad \text{Equation 4-12}$$

The positions of the center of mass for the individual links in base coordinate system are calculated by applying the transformation matrixes to the centre of mass for each link. Figure 4-10 shows the diagram for the total center of mass calculation.

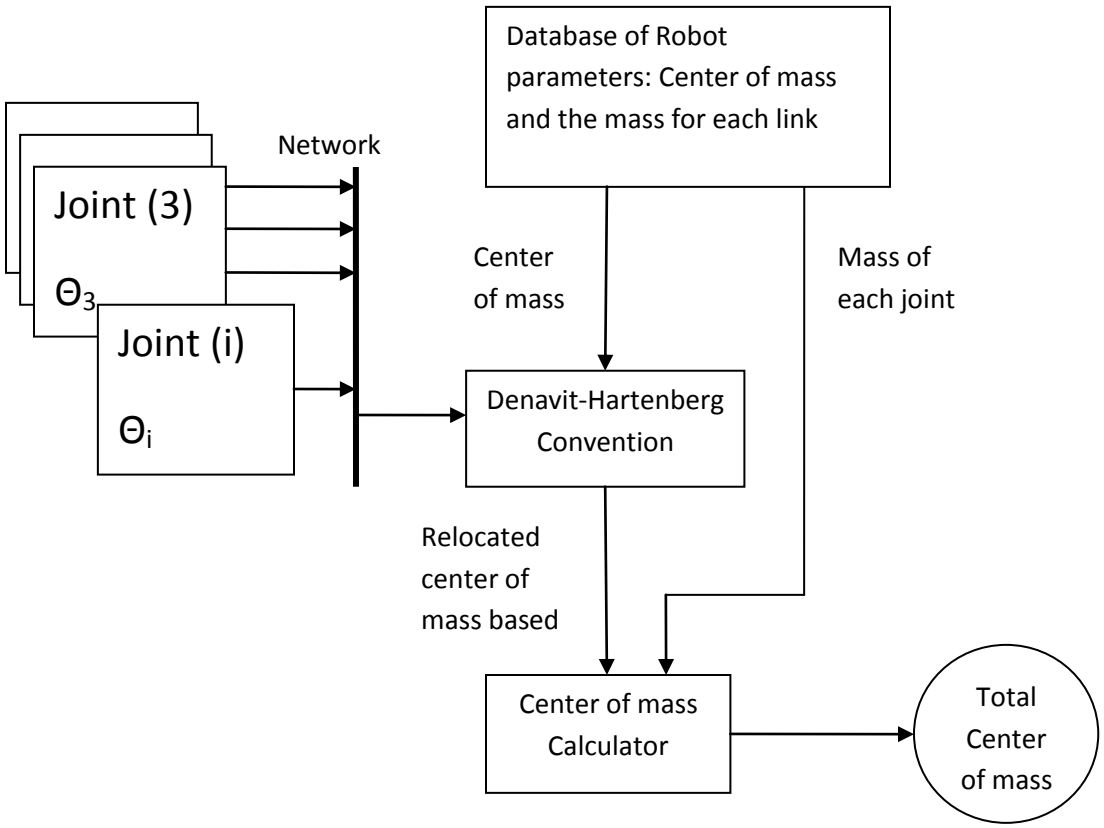


Figure 4-9 : Center of mass calculation

The following calculation is optimized in order to reduce the load of the processing in the central controller.

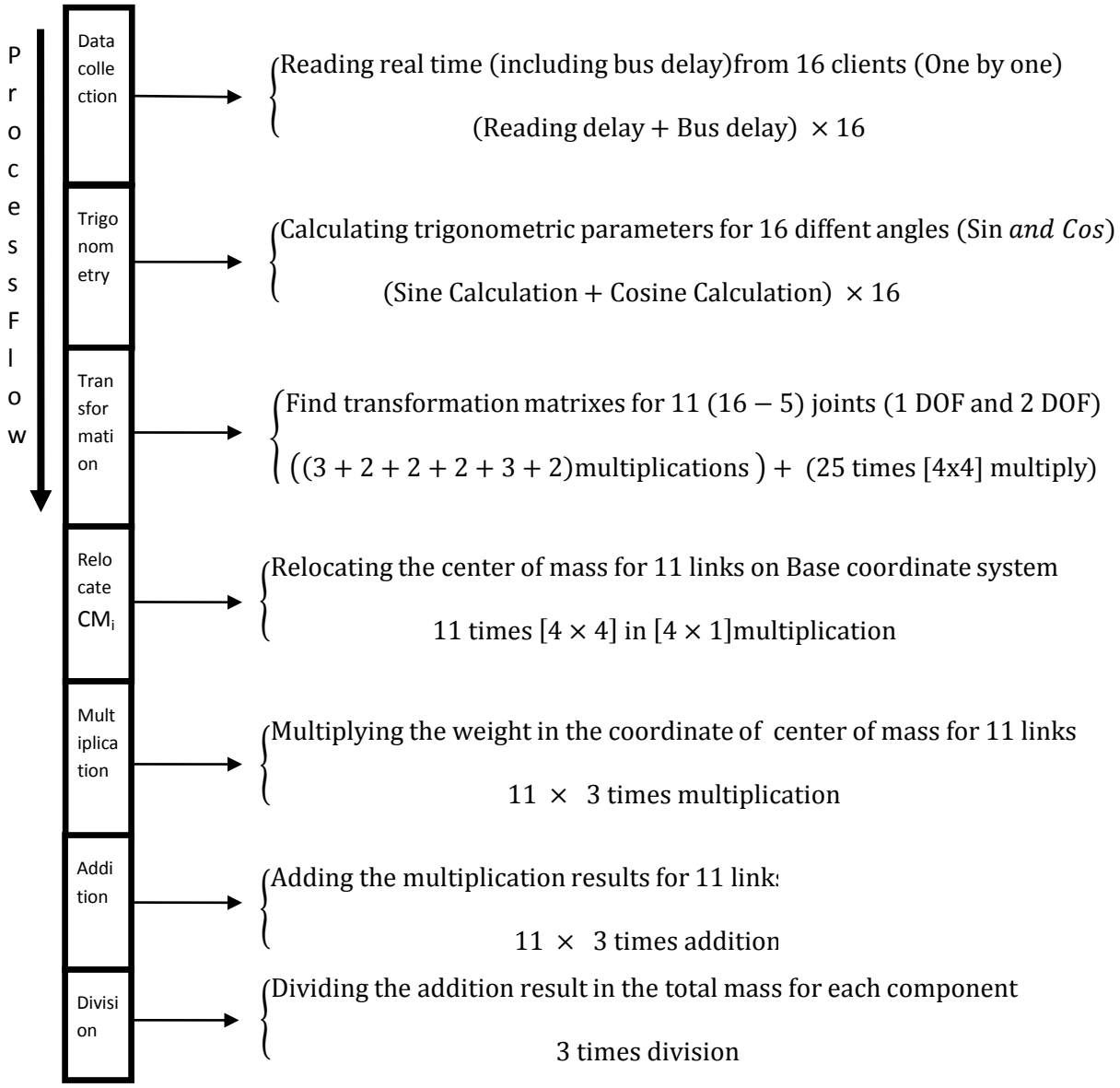


Figure 4-10 : Center of mass calculation with the traditional method

Regarding the structure of Archie, the links are named as it is shown in figure 4-11. Table 4-2 shows the weight and the position of the center of mass for each link based on their coordinate system.

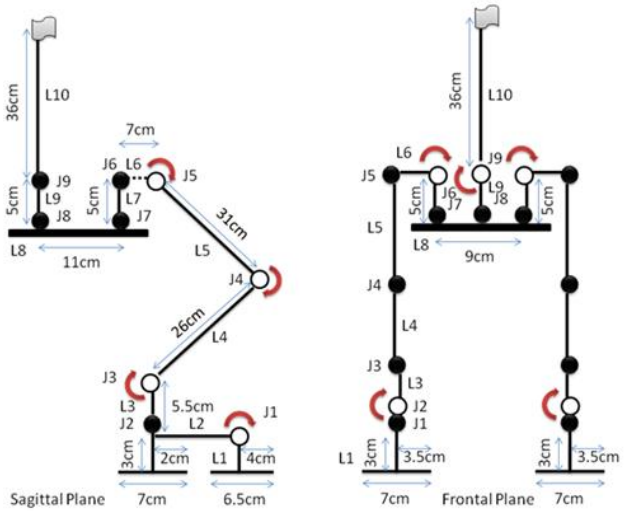


Figure 4-11: Sagittal and frontal view of the links and joints of Archie

Link Name	Mass	Center of mass X	Center of mass Y	Center of mass Z
L1	0.125kg	0mm	-53mm	-4mm
L2	0.111kg	0mm	-4mm	-26mm
L3	0.008kg	0mm	0mm	-5mm
L4	0.075kg	28mm	0mm	-130mm
L5	0.131kg	98mm	0mm	-155mm
L6	0.048kg	58mm	23mm	0mm
L7	0.049kg	0mm	0mm	25mm
L8	0.346kg	75mm	70mm	-15mm
L9	0.049kg	0mm	0mm	22mm
L10	2.992kg	0mm	33mm	249mm

Table 4-2: Link's masses and position of center of mass for each joint based on its coordinate

4.3.6. Moment of inertia calculation

Moment of inertia is the rotational analog of the mass. According to the mechanical structure of a humanoid robot, all the movements are based on revolute joints. Thus, finding the moment of inertia is necessary for modeling the joints. For a rigid object of N point masses m_i, the moment of inertia tensor is given by:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \qquad \text{Equation 4-13}$$

Where the elements defined for Cartesian coordinates (x_i, y_i, z_i) with the origin at the center of the mass are:

$$I_{xx} = \sum_{i=1}^{N} m_i({y_i}^2 + {z_i}^2) \qquad \text{Equation 4-14}$$

$$I_{yy} = \sum_{i=1}^{N} m_i({x_i}^2 + {z_i}^2) \qquad \text{Equation 4-15}$$

$$I_{zz} = \sum_{i=1}^{N} m_i({x_i}^2 + {y_i}^2) \qquad \text{Equation 4-16}$$

$$I_{xy} = I_{yx} = -\sum_{i=1}^{N} m_i x_i y_i \qquad \text{Equation 4-17}$$

$$I_{xz} = I_{zx} = -\sum_{i=1}^{N} m_i x_i z_i \qquad \text{Equation 4-18}$$

$$I_{yz} = I_{zy} = -\sum_{i=1}^{N} m_i y_i z_i \qquad \text{Equation 4-19}$$

The diagonal elements in the inertia tensors (I_{xx}, I_{yy}, I_{zz}) are called the moments of inertia while the rest of the elements are called the products of inertia. I_{xx} relates to the moment of inertia around the x-axis when the objects are rotated around the x-axis, I_{xy} relates to the moment of inertia around the y-axis when the objects are rotated around the x-axis, etc.

The inertia tensor has only six independent coordinates, three diagonal elements and three off-diagonal. There are also three other elements which are dependent on the location and orientation of the local reference frame. It is always possible for a rigid body, to align a local reference frame in which the mass of the body is evenly distributed around the axes. In that case, the inertia tensor becomes purely diagonal:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \qquad \text{Equation 4-20}$$

These coordinate's axes are called the principal axes. Information about the values of inertia for every link is derived from equations 4-14, 4-15 and 4-16 which are given in Table 4-3:

Link	Inertia Matrix I_{xx}	Inertia Matrix I_{yy}	Inertia Matrix I_{zz}
L1	7.086e-005kgm^2	1.345e-004kgm^2	1.694e-004kgm^2
L2	1.601e-004kgm^2	1.107e-004kgm^2	8.256e-005kgm^2
L3	4.044e-006kgm^2	1.251e-006kgm^2	4.937e-006kgm^2
L4	5.608e-004kgm^2	5.857e-004kgm^2	7.182e-005kgm^2
L5	0.002kgm^2	0.002kgm^2	7.166e-005kgm^2
L6	3.679e-005kgm^2	6.507e-005kgm^2	8.396e-005kgm^2
L7	4.577e-005kgm^2	5.084e-005kgm^2	2.541e-005kgm^2
L8	9.895e-004kgm^2	9.418e-004kgm^2	0.001kgm^2
L9	4.577e-005kgm^2	5.084e-005kgm^2	2.541e-005kgm^2
L10	0.028kgm^2	0.038kgm^2	0.017kgm^2

Table 4-3: Values of principal axes corresponding to every link of the robot

The perpendicular distance of the axes can be found by unifying the coordinate system of the joints using the based coordinate system.

To calculate the moment of inertia for each link in a specific point (around a specific axis), the parallel axis theorem (Huygens-Steiner Theorem) is used. Regarding the parallel axis theorem, the moment of inertia of an object about any axis can be found using the moment of inertia about a parallel axis (parallel to the aimed axis) through the object's center of mass and the perpendicular distance between the axes. Figure 4-12 shows the following theorem.

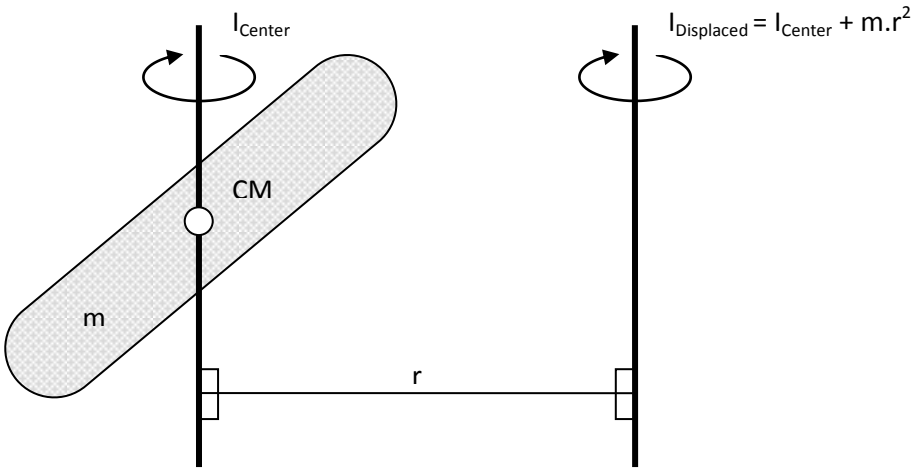


Figure 4-12: Parallel axis theorim

The parallel axis theorem can be generalized to calculate the new displaced tensor of inertia (J_{ij}) as in equation 4-21.

$$J_{ij} = I_{ij} + m\left(r^2\delta_{ij} + r_i r_j\right) \qquad \text{Equation 4-21}$$

Where: I_{ij} is the principal moment of inertia, calculated over the object's center of mass.

Using the following theorem the moment of inertia reflected on each joint can be calculated based on the position and situation of the other links and joints of the robot.
Using the Denavit-Hartenberg model, the central controller can find the rotation and the distance of each link (the center of mass of each link) to each joint and can calculate the reflected moment of inertia from that link. An accumulation of all the links on a certain joint (based on the situation of the links) gives the reflected moment of inertia to that joint.
The reflected moment of inertia calculation for an arbitrary joint on a specific joint is based on two parameters. The first parameter is the distance between the center of mass of the link and the desired axis (which is used in the parallel axis theorem), and the second is the rotation of the link (described using Euler angles). The rotation parameters are used to calculate the moment of inertia for a specific object (link) about an arbitrary axis.
Regarding to table 4-3 the principal moment of inertia is calculated over the Cartesian axes, the moment of inertia for and arbitrary axis is calculated using the following equation:

$$I_{Arbitrary} = I_{xx}.|\cos(\psi).\cos(\theta)| + I_{yy}.|\sin(\psi).\cos(\theta)| + I_{zz}.|\sin(\theta)| \qquad \text{Equation 4-22}$$

Where:

ψ: Angle between the reflection of the arbitrary axis in XY plane and the X axis.

Θ: Angle between the arbitrary axis and XY plane.

4.4. Joint's motion controller

In this section the joint controllers used in the robot are described. The model is based on considering the load (links connected to the joints) as a constant point mass (Center of mass of each link) at a specific distance from the joint; the distance increases on the worst case angle related to each joint. The worst case angle arises from the gravity effect that affects the links differently on diverse angles.

4.4.1. Modeling and controlling the joints

The controller used for controlling the joints of the robot is based on a velocity controller. The velocity controller is commanded using a velocity trajectory planner in order to control the position of the joint.

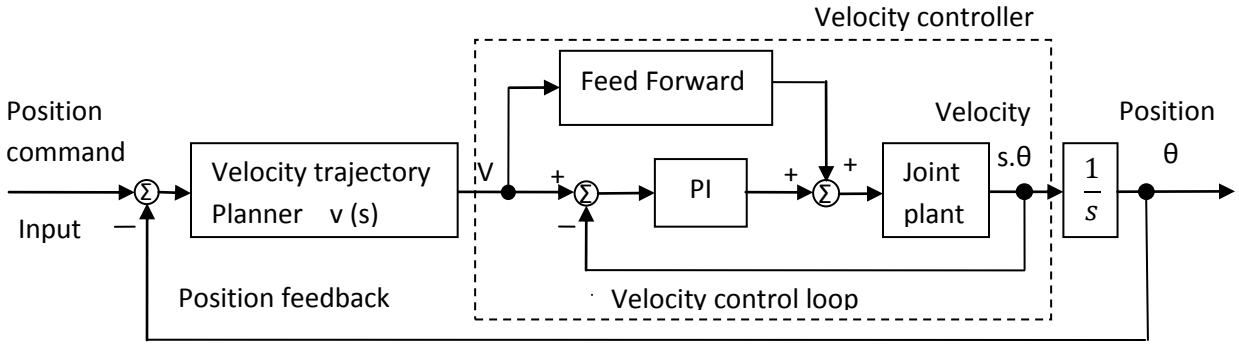


Figure 4-13: Control loop used for the joints

The controller calculates a velocity trajectory to achieve the desired position for the output shaft. The velocity trajectory is the trapezoidal shown in figure 4-14.

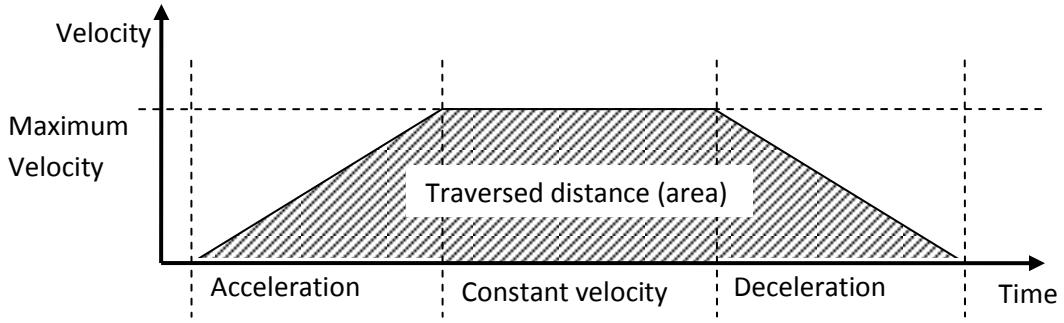


Figure 4-14: Velocity trajectory and position

4.4.2. Model of the joint plant

The joints in Archie are based on a DC motor (i.e., either brushed or brushless). The following is an electrical schematic of a DC motor and the mechanical load attached to it.

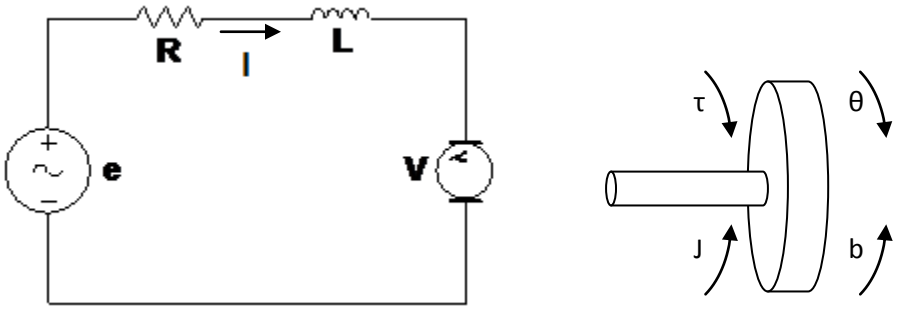


Figure 4-15: Simplified schematic of a DC motor

The effect of the harmonic gear can be modeled as follows. Although it has the same effect as a simple gear, the actual robot joint is based on harmonic gear.

Using Newton's law and Kirchhoff's law we will have the following equations:

From Newton's law: $\tau_m - \frac{\tau_l}{N} = J\ddot{\theta_m} + b\dot{\theta_m} \quad \rightarrow \quad K.i - \frac{\tau_l}{N} = J\ddot{\theta_m} + b\dot{\theta_m}$ Equation 4-23

From Kirchhoff's law: $L\frac{di}{dt} + Ri = V - e \quad \rightarrow \quad L\frac{di}{dt} + Ri = V - K\dot{\theta_m}$ Equation 4-24

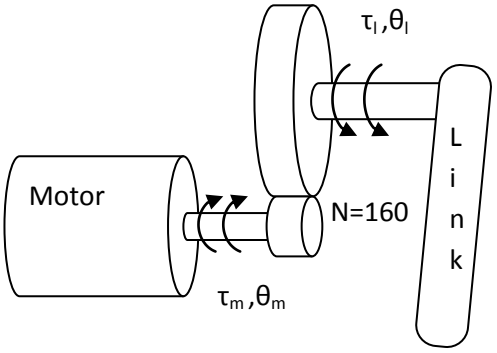


Figure 4-16: Motor gear and the link model

The J is the combination of the moment of inertia of the motor's rotor and the moment of inertia reflected from the load on the gear's input:

$$J = J_{rotor} + J_{gearbox} + J_{load} \cdot \left(\frac{1}{N}\right)^2 \quad \text{Equation 4-25}$$

The friction on the load can be calculated using the following formula:

$$b = b_{rotor} + b_{load} \cdot \left(\frac{1}{N}\right)^2 \quad \text{Equation 4-26}$$

Where J is the moment of inertia and b is for the friction losses.

Using Laplace transformation the above equations can be expressed as:

$$(s.L + R).I(s) = E(s) - s.K.\theta_m(s) \quad \text{Equation 4-27}$$

$$(J.S^2 + b.s).\theta_m(s) = K.I(s) - \frac{\tau_l(s)}{N} \quad \text{Equation 4-28}$$

The transfer function, between the input voltage $E(s)$and the motor's rotor position $\theta_m(s)$is:

$$\frac{\theta_m(s)}{E(s)} = \frac{K}{s\big((s.L + R)(J.s + b) + K^2\big)} = \frac{K}{s^3 J.L + s^2(J.R + b.L) + s(R.b + K^2)}$$

Equation 4-29

In addition the transfer function, between the load torque $\tau_l(s)$ and motor position $\theta_m(s)$ is:

$$\frac{\theta_m(s)}{\tau_l(s)} = -\frac{1}{N}\frac{(s.L+R)}{s\big((s.L+R)(J.s+b)+K^2\big)} \quad \text{Equation 4-30}$$

The links are affected by the gravity force which is related to the angles of the links of the robot.

4.4.3. Inclusion of gravity force

The gravity is affecting all the links depending on their angles of slant. Therefore, the following forces are not constant. The following force is reflected as a negative torque in the system and it is decreasing from the motor output torque.

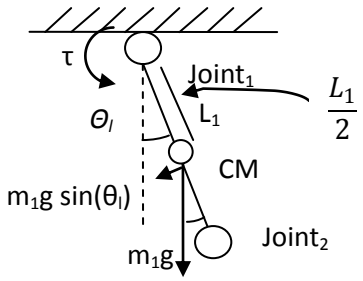


Figure 4-17: Gravity effect on a link

Thus, the following equations can be written as equation 4-31.

$$\tau = J\ddot{\theta}_l + b\dot{\theta}_l + m.g.\frac{L_1}{2}.\sin(\theta_l) \qquad \text{Equation 4-31}$$

The term $\sin(\theta_l)$ causes non-linearity in the above equation; therefore linearization about the maximum angle, θ_{max} using Taylor series can be applied. The linearization function is:

$$f(\theta_l) = f(\theta_{max}) + \frac{df}{d\theta}|_{\theta=\theta_{max}} \frac{(\theta_l - \theta_{max})}{1!} + \frac{d^2f}{d\theta^2}|_{\theta=\theta_{max}} \frac{(\theta_l - \theta_{max})^2}{2!} + \cdots$$

Equation 4-32

Regarding the small effect of θ_l about θ_{max} , only the first two terms will be considered.

$$f(\theta_l) - f(\theta_{max}) = \frac{df}{d\theta}|_{\theta=\theta_{max}} \frac{(\theta_l - \theta_{max})}{1!} \qquad \text{Equation 4-33}$$

$$\delta f(\theta_l) = p|_{\theta=\theta_{max}}\, \delta\theta_l \qquad \text{Equation 4-34}$$

The following equation is the linear approximation where p is the slope at the θ_{max} .

Using the above equation the $\sin(\theta_l)$ can be substituted as following:

$$\sin(\theta_l) - \sin(\theta_{max}) = \cos(\theta_{max}).(\theta_l - \theta_{max}) \qquad \text{Equation 4-35}$$

$$\sin(\theta_l) = \sin(\theta_{max}) + \cos(\theta_{max}).\delta\theta_l \qquad \text{Equation 4-36}$$

Using the equations 4-35 and 4-36 the gravity effect can be written as in equation 4-37.

$$\tau_G = J.\delta\ddot{\theta}_l + b.\delta\dot{\theta}_l + m.g.\frac{L_1}{2}(\sin(\theta_{max}) + \cos(\theta_{max}).\delta\theta_l) \qquad \text{Equation 4-37}$$

Using Laplace transformation of equation 4-37:

$$\tau(s) = J.S^2\delta\theta_l + b.S.\delta\theta_l + m.g.\frac{L_1}{2}\sin(\theta_{max}) + m.g.\frac{L_1}{2}.\cos(\theta_{max}).\delta\theta_l \qquad \text{Equation 4-38}$$

Using substitution the equation 4-38 in the equation 4-29:

$$E(s) = \frac{R}{K}JS^2.\delta\theta_m + \frac{R}{K}bS.\delta\theta_m + \frac{R}{K}mg\frac{L_1}{2}\sin(\theta_{max}) + \frac{R}{K}mg.\frac{L_1}{2}.\cos(\theta_{max}).\delta\theta_m + K.S.\delta_m$$

Equation 4-39

The equation 4-39 contains a constant term $\frac{R}{K}mg\frac{L_1}{2}\sin(\theta_{max})$ which represents the gravitation effect. The plant model is shown in block diagram figure 4-18.

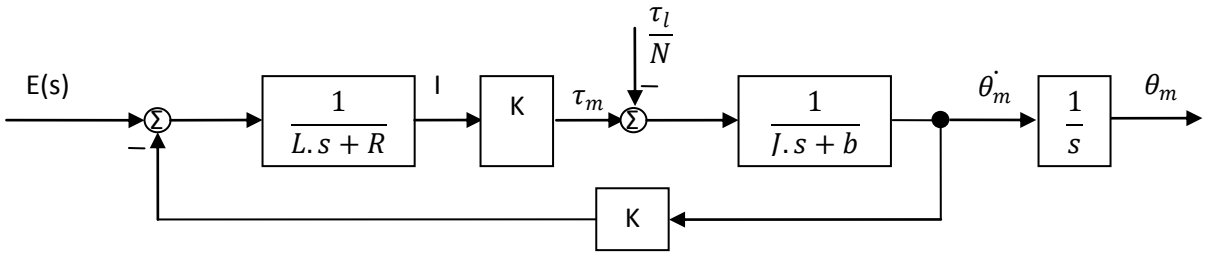


Figure 4-18: Model of the joint plant

4.4.4. Model order reduction

The resulted transfer function shown above is a second order, type zero system. The system includes a mechanical pole and an electrical pole. The mechanical pole is slower than the electrical pole, which means the time constant of the mechanical pole is larger than the time constant of the electrical pole. In an open loop the electrical pole moves greatly to the left side of the S-Plane. Using this outcome the transfer function can be reduced to the first order system with the dominant mechanical pole. Regarding the real values from the actual design in Archie's joints:

$$Electrical\ pole\ time\ constant = \frac{L}{R} = \frac{0.573\times10^{-3}}{0.978} = 5.85\times10^{-4} \qquad \text{Equation 4-40}$$

$$Mechanical\ pole\ time\ constant = \frac{J}{b} = \frac{0.0078}{30.46\times10^{-3}} = 2.56\times10^{-1} \qquad \text{Equation 4-41}$$

The following comparison shows the large difference between the time constant of the Electrical pole and the Mechanical pole. The transfer function between the link's angle and the motor's voltage can be simplified as follows:

$$\frac{\theta_m(s)}{E(s)} = \frac{1}{s}\frac{K}{((L.s+R)(J.s+b))+K^2} = \frac{1}{s}\frac{\frac{K}{R}}{\frac{L}{R}Js^2+\frac{L}{R}bs+Js+b+\frac{K^2}{R}} = \frac{1}{s}\frac{\frac{K}{R}}{Js+b+\frac{K^2}{R}}$$

Equation 4-42

The simplified model can be described in a block diagram as follows:

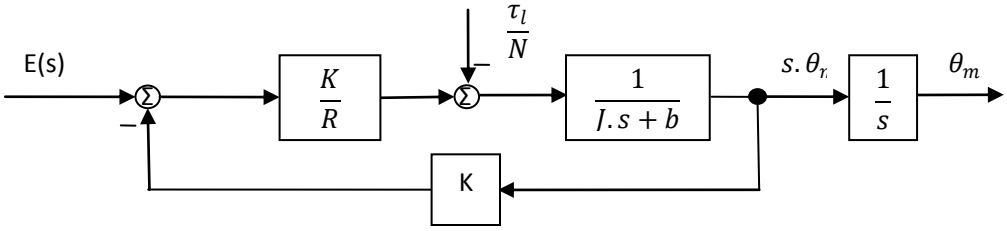


Figure 4-19: Simplified model of the joint plant

4.4.5. Controlling the joint

The model of the joint controller is presented above. The controller can move the joint to a desired position based on the requested traversing velocity. The position and traversing velocity are issued by the central controllers and sent to the individual controllers to achieve the overall control. Controlling the joints is accomplished by using a proportional-integral (PI) controller with a feed forward cancellation.

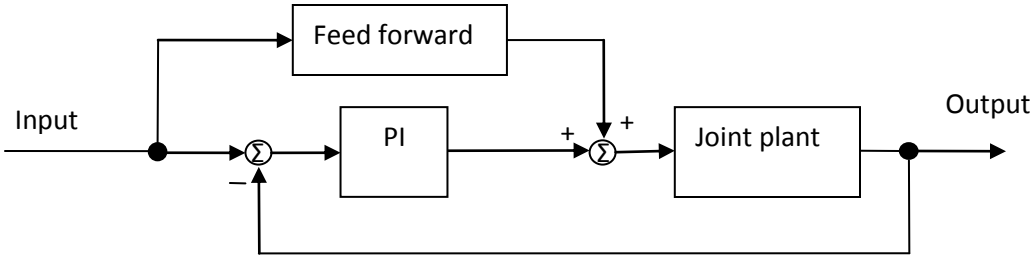


Figure 4-20: The control scheme of one joint

The PI compensator is used to eliminating the steady state error by adding a pole in the origin of the s-plan. Adding this pole will increase the type of the system. The output of the PI block is the summation of the coefficient of the error and a coefficient of the integral of the error over time.

$$f(t) = K_p e(t) + \frac{K_p}{T}\int_0^t e(t)\,dt \qquad \text{Equation 4-43}$$

Where K_p, is the proportional gain and $K_i = \frac{K_p}{T}$ is the integral gain, which the T is the control loop period. The following equation with Laplace transformation is:

$$F(s) = K_p E(s) + \frac{K_p}{T}\frac{E(s)}{s} \qquad \text{Equation 4-44}$$

From which:

$$\frac{F(s)}{E(s)} = K_p\left(1 + \frac{1}{T_s}\right) = K_p\left(\frac{T_s+1}{T_s}\right) = K_p\left(\frac{s+\frac{1}{T}}{s}\right) \qquad \text{Equation 4-45}$$

Using root locus technique the position of the PI zero can be replaced adequately close to the PI pole that is located in the origin of the s-plan. Using the following technique the order of the system will be increased while the root locus will remain unaffected. Since the zero is located at $-\frac{1}{T}$, the first order mechanical pole will move the left side of the s-plane for different values of K_p.

According to the first order approximation, K_p can move the first order pole to infinity. The maximum value that K_p can take in order for the response of the system to still be over-damped is the length from the first order pole to the breakaway point between the electrical and mechanical poles. Furthermore the closed loop pole enters the complex plane and the response becomes under damped with an overshoot.

The feed forward path is an inverse model of the joint model. Regarding to the resulting transfer function for the joint:

$$\frac{\dot{\theta_m}(s)}{E(s)} = \frac{K}{s^2.J.L+s.(J.R+b.L)+(R.b+K^2)} \qquad \text{Equation 4-46}$$

The inverse of the function above is:

$$\frac{E(s)}{\dot{\theta_m}(s)} = \frac{s^2.J.L+s(J.R+b.L)+(R.b+K^2)}{K} \qquad \text{Equation 4-47}$$

The block diagram description of the feed forward block is shown in figure 4-21:

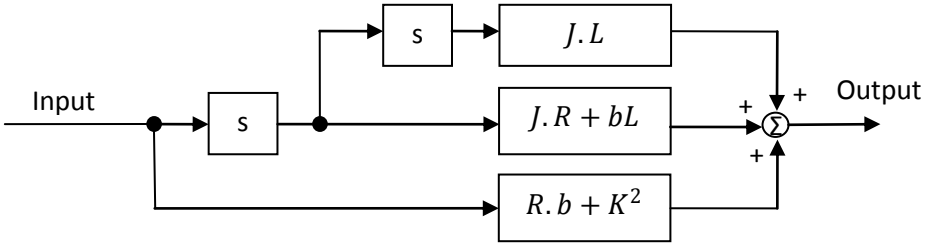


Figure 4-21: Feed forward block diagram

The main problem of the joint controllers is the instability in the model of the joint. The instability is rooted in the variable moment of inertia (J) reflected in the joint from the links and the gravity effect. To deal with this issue, the controller parameters can be set on a value to achieve stability in the controller. The second option is using pre-calculated parameters from the central controller that are taken from the kinematics model of the system. The gravity effect is calculated based on the hip on the horizontal position. Each deflection in the hip position is sensed using the inertial measurement unit (IMU) in order to make it possible for the robot to walk on non-even terrains still without sole pressure sensors.

4.4.6. Derivation of the joint controller

In this section, the resulted motion controller is prepared for one of the joints of the robot. For instance, the lateral hip joint is modeled during the single support phase for the swinging leg. The following calculations can be extended for the other joints of the robot using the same method. Figure 4-22 shows the swinging leg attached to the trunk using the hip joint which is acting like the classic pendulum problem.

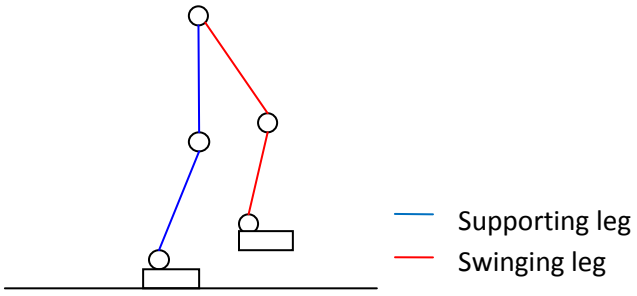


Figure 4-22: Single support phase

In figure 4-23 the description of the swinging leg is shown by the mass points connected to each other using weightless links.

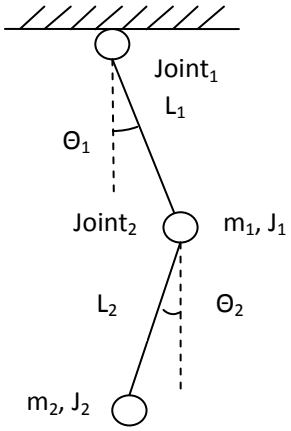


Figure 4-23: Double pendulum model

Regarding to the motor (EC-45, Maxon-motor, 2002) datasheet and mechanical system specifications, the constant parameters of each joint in Archie are as follows:

L=0.573mH R=0.978Ω K=33.5 mNm/A J_{motor}=135gcm^2/s^2 J_{gear}=320gcm^2/s^2 b=0.28mNms

4.4.7. Range of the moment of inertia deviation

For instance, the moment of inertia for the lateral hip in swinging phase is obtained from the equation 4-51.

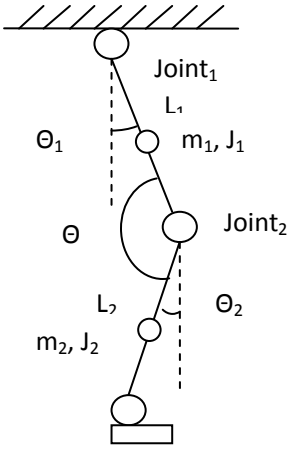


Figure 4-24: Swinging leg model

$$J = \sum J_{i,1} = J_{1,1} + J_{1,2} \qquad \text{Equation 4-48}$$

$$J_{1,1} = J_{1,yy} + m_1.(\frac{l_1}{2})^2 \qquad \text{Equation 4-49}$$

$$J_{1,2} = J_{2,yy} + m_2.\left(\frac{\sqrt{{l_1}^2+\left(\frac{l_2}{2}\right)^2-2.l_1.\frac{l_2}{2}.cos\theta}}{2}\right)^2 \qquad \text{Equation 4-50}$$

$$J = J_{1,yy} + J_{2,yy} + m_1.\left(\frac{l_1}{2}\right)^2 + \frac{m_2.\left({l_1}^2+\left(\frac{l_2}{2}\right)^2-l_1.l_2.cos\theta\right)}{4} \qquad \text{Equation 4-51}$$

The value of the moment of inertia (J) is different for each joint and is described in a range. As a result, the value of K_p should be chosen appropriately. The suitable values of K_p can be found by either tuning the control loop, or modeling the joint and finding the value using simulation and classical tuning methods.

A derivation from the moment of Inertia for the swinging leg of Archie is calculated as follows.

$$J1_{yy}=0.000857kgm^2 \quad , \quad J2_{yy}=0.002kgm^2$$

$$J = 0.0677 + 0.014cos\theta \qquad \text{Equation 4-52}$$

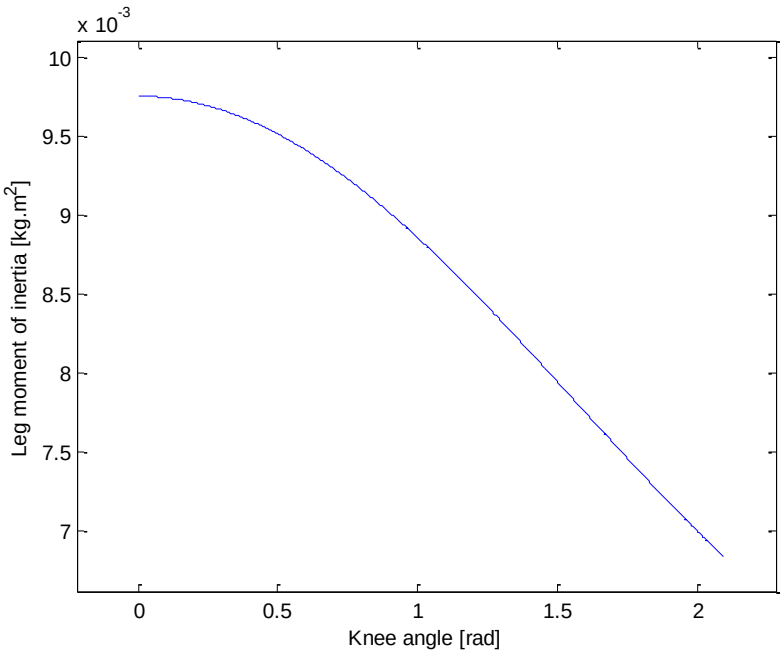


Figure 4-25: Moment of inertia in lateral hip joint on knee angle change for swinging leg

In order to provide stability for the system and by using the resulted range for the moment of inertia, the parameters of the control loop can be chosen. To calculating the moment of inertia for each joint the situation of the joint should be considered. For instance, the knee joint faces different reflection of moment of inertia from the attached load by switching from the swinging phase to the supporting phase. The same calculation should be derived for all joints of the robot. Table 4-4 shows the range of the moment of inertia for each joint of the robot.

Joint	Minimum reflected moment of inertia (swinging) [Kgm^2]	Maximum reflected moment of inertia (swinging) [Kgm^2]	Minimum reflected moment of inertia (supporting) [Kgm^2]	Maximum reflected moment of inertia (supporting) [Kgm^2]
J1 (toe)	1.3452e-004	1.3452e-004	1.8541e-001	1.8648e-001
J2 (F Ankle)	3.1321e-004	3.1849e-004	2.2336e-001	2.7624e-001
J3 (L Ankle)	1.0413e-003	1.0466e-003	1.8385e-001	1.5083e-001
J4 (Knee)	1.1414e-002	1.2789e-002	1.0879e-001	1.3151e-001
J5 (L Hip)	5.3704e-002	8.1630e-002	2.6586e-001	2.8691e-001
J6 (F Hip)	6.1736e-002	1.0158e-001	1.7406e-001	1.9636e-001
J7 (T Hip)	5.0974e-004	1.3661e-001	1.9005e-001	3.2689e-001
J8 (T Torso)	3.8050e-002	6.2286e-002	3.8050e-002	6.2286e-002
J9 (F Torso)	1.2494e-001	1.2494e-001	1.2494e-001	1.2494e-001

Table 4-4: Reflected moment of inertia range for each joint during swinging and supporting phase

4.4.8. Range of the gravity effect deviation

The gravity effect should be calculated for each joint in the same way. The result from the gravity effect calculation can be represented by the representative mass and the length of the lever. The mass reflected in each joint is constant for each phase; however, the length of the lever changes on different poses of the other joints. For instance, in this section the calculations for the gravitational effect that is reflected to the hip joint are presented.

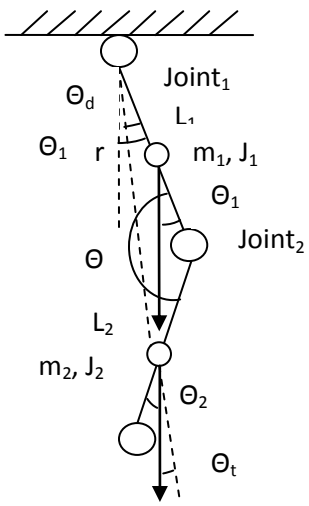


Figure 4-26: Gravity effect in the double pendulum

$$\tau_G = \tau_{g,1} + \tau_{g,2}$$ Equation 4-53

$$\tau_{g,1} = m_1.g.\frac{l_1}{2}.cos\theta_1$$ Equation 4-54

$$\tau_{g,2} = m_2.g.cos\theta_t\sqrt{{l_1}^2 + \left(\frac{l_2}{2}\right)^2 - 2.l_1.\frac{l_2}{2}cos\theta}$$ Equation 4-55

$$\theta_t = \theta_1 - \theta_d$$ Equation 4-56

Regarding to law of cosines θ_d is calculated as equation 4-57.

$$\theta_d = Arccos\left(\frac{2.l_1 - l_2.cos\theta}{2\sqrt{{l_1}^2 + \left(\frac{l_2}{2}\right)^2 - l_1.l_2.cos\theta}}\right)$$ Equation 4-57

After further calculations, the effect of gravity is factored into the equation as described in equation 4-58:

$$\tau_G =$$

$$m_1.g.\frac{l_1}{2}.cos\theta_1 + m_2.g.cos\left(\theta_1 - Arccos\left(\frac{2.l_1 + l_2.cos\theta}{2\sqrt{{l_1}^2 + \left(\frac{l_2}{2}\right)^2 - l_1.l_2.cos\theta}}\right)\right)\sqrt{{l_1}^2 + \left(\frac{l_2}{2}\right)^2 - l_1.l_2.cos\theta}$$

Equation 4-58

For Archie, the gravitational effect would be as in equation 4-59.

$$m_1 = 0.131kg, m_2 = 0.075kg, l_1 = 0.30m, l_2 = 0.26m, g = 9.8$$

$$\tau_G = (.193).cos\theta_1 + (.735)cos\left(\theta_1 - Arccos\left(\frac{.30 - (.13).cos\theta}{\sqrt{(.107) - (.078).cos\theta}}\right)\right)\sqrt{.107 - (.078).cos\theta}$$

Equation 4-59

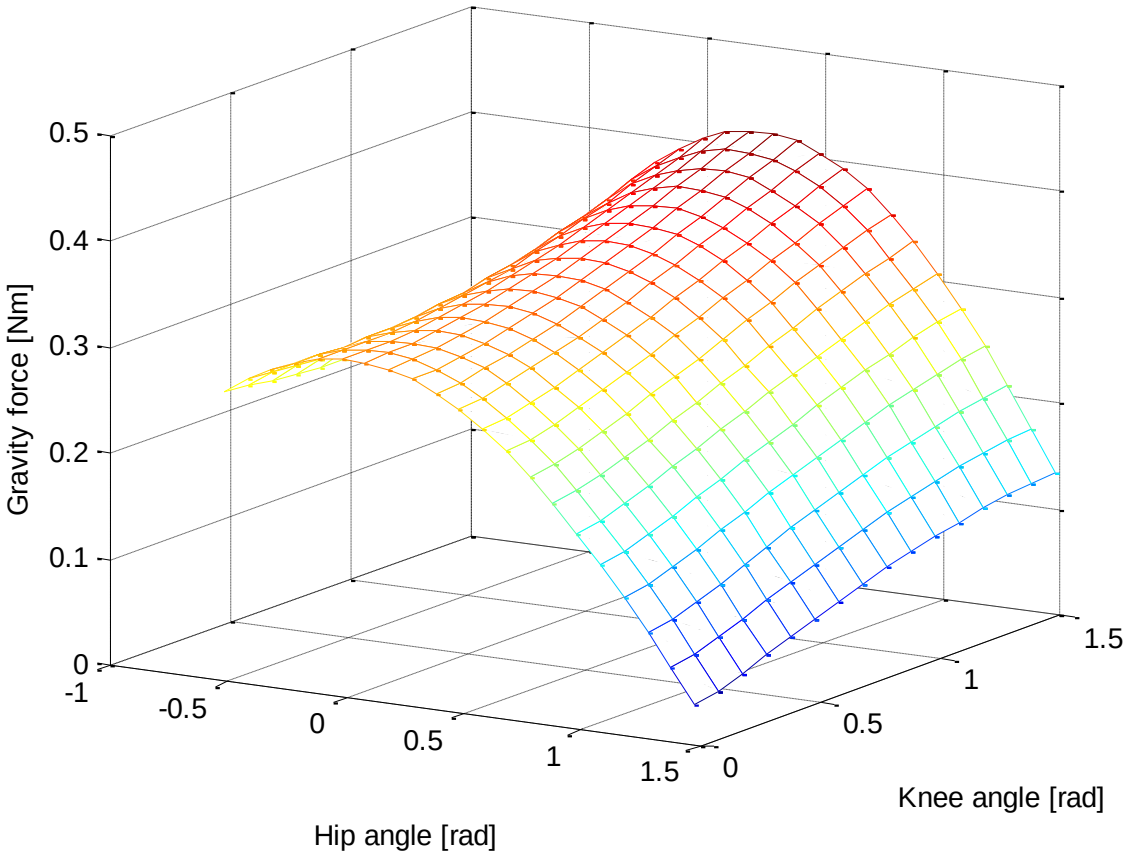


Figure 4-27: Gravity force for swinging leg on knee and hip angle changing

4.5. Digital controller

The digital controller is a type of controller that uses a digital computer (processor) to act as a system controller. Since a digital computer is a discrete system, the Laplace transformation cannot be used, and the Z-transformation is used instead of it.

Using a digital controller brings some benefits to the system. Some of these benefits are vital for a humanoid robot. An example of such benefit is the opportunity to change the parameters of the controllers during operation using software methods. Since in a humanoid robot, the reflected load and the moment of inertia for the joints are variable, the new information should be returned to the controllers in order to adapt to the system for the new load properties.

The first step in designing the digital controller system is converting the continuous transfer function to a discrete transfer function. The continuous transfer function for the joint is:

$$\frac{\dot{\theta_m}(s)}{E(s)} = \frac{K}{s^2.J.L+s.(J.R+b.L)+(R.b+K^2)}$$ Equation 4-60

Where the values described for the motor are as following:

L=0.573mH R=0.978Ω K=33.5 mNm/A J_{motor}=135 gcm^2 b=30.46mNms

Regarding table 4-4 and the total reflected moment of inertia on the joint (Equation 4-51) the range is:

$$J_{max} = 1.35 \times 10^{-5} + 8.16 \times 10^{-2}.\left(\frac{1}{160}\right)^2 = 5.235 \times 10^{-4}$$ Equation 4-61

$$J_{min} = 1.35 \times 10^{-5} + 5.37 \times 10^{-2}.\left(\frac{1}{160}\right)^2 = 3.491 \times 10^{-4}$$ Equation 4-62

$$J_{average} = \frac{J_{max}+J_{min}}{2} = \frac{5.235\times10^{-4}+3.491\times10^{-4}}{2} = 4.363 \times 10^{-4}$$ Equation 4-63

Because of the instability of the system (variable load parameters) the average of the moment of inertia, reflected on the joint is used for the calibration. The transfer function is:

$$H(s) = \frac{\dot{\theta_m}(s)}{E(s)} = \frac{3.35}{(7.907\times10^{-7})s^2+(1.366\times10^{-3})s+0.1396}$$ Equation 4-64

With respect to the control loop time period ($T_s = \frac{1}{350Hz} = 0.00285ms$) the continuous transfer function can be transformed to discrete transfer function (Z-transform) which is:

$$H(z) = \frac{\dot{\theta_m}(z)}{E(z)} = \frac{(5.174)z+1.19}{z^2-(0.742)z+7.194\times10^{-3}}$$ Equation 4-65

The following transfer function has the step response shown in figure 4.28:

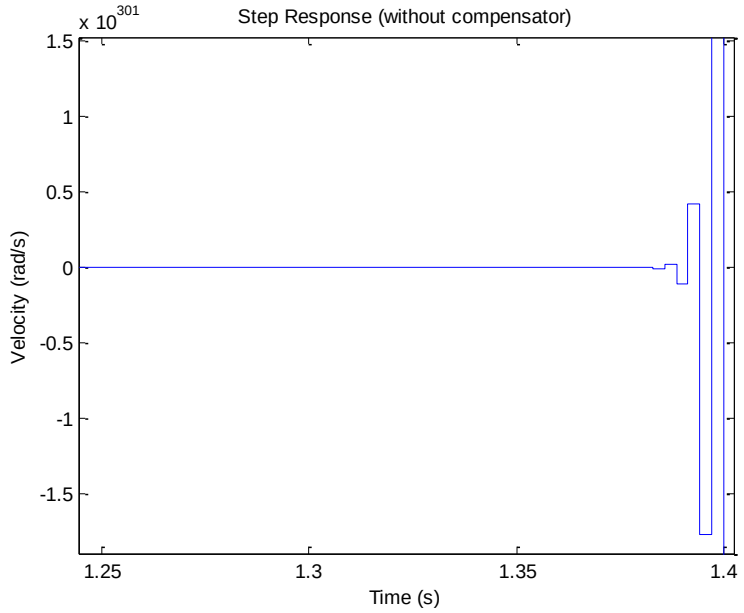


Figure 4-28: Step response of the control loop without compensator (simulation)

The system is unstable and starts to resonance after 1.38 seconds. Figure 4-29 shows the Root locus diagram of the following system.

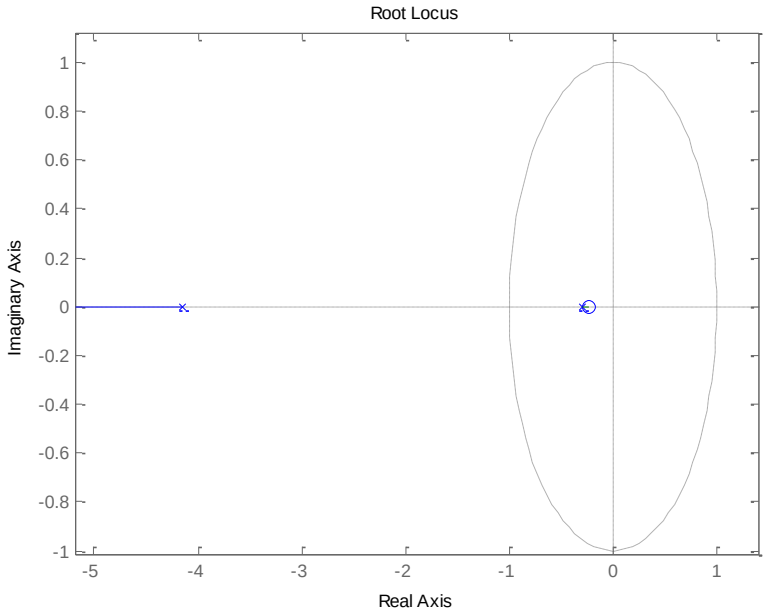


Figure 4-29: Root locus diagram of the control loop without compensator (simulation)

In the Root locus diagram shown in figure 4-29, one pole is placed outside of the unique circle which causes the instability in the system.

As shown in figure 4-13 and the controller scheme, the velocity of the joint is affected by a Proportional-integral (PI) compensator. Since the controller is based on discrete time system; the compensator should be transformed from continuous to discrete time system.

$$K_P + \frac{K_I}{s} \qquad \text{Equation 4-66}$$

Using the Bilinear Transformation (Tustin method which uses ($s = \frac{2}{T_s}.\frac{z-1}{z+1}$) substitution):

$$K_P + K_I.\frac{T_s}{2}.\frac{z+1}{z-1} \qquad \text{Equation 4-67}$$

After using the tuning algorithm the values of K_P and K_I were found 39 and 4578 respectively. As shown in the root locus diagram of the control system (including the PI compensator) the gain (K_P) has an utmost. By crossing the K_P utmost, one or more poles will move to the right side of the s-plan which causes instability for the control loop (shown in figure 4-30).

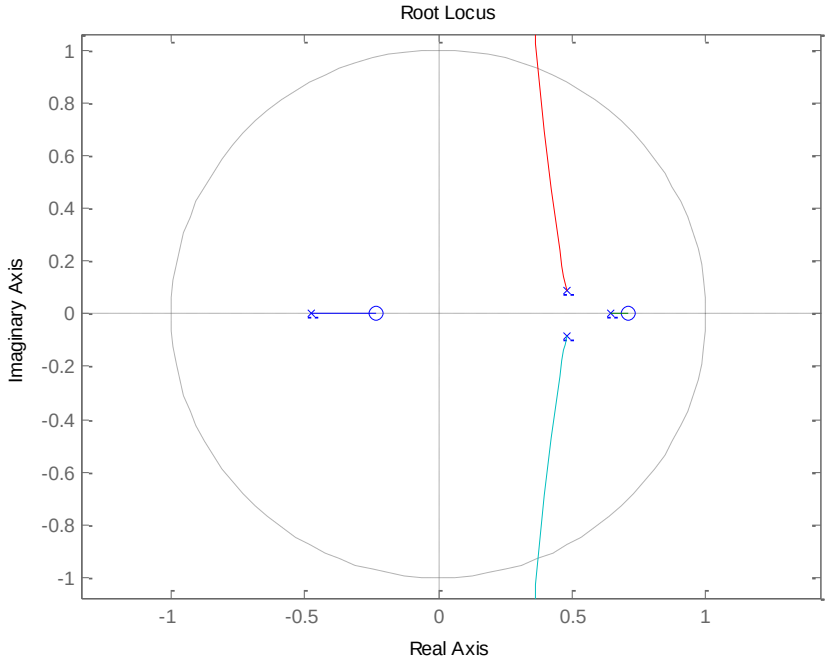


Figure 4-30: Root locus diagram for tuned compensated control loop (simulation)

Figure 4-30 shows the location of the pole inside the unique circuit, although two imaginary poles are applying an utmost value for the loop gain to be located in the unique circuit. Figure

4-31 depicts the step response used to test the control loop by regarding the maximum value of the used motor (500rad/s).

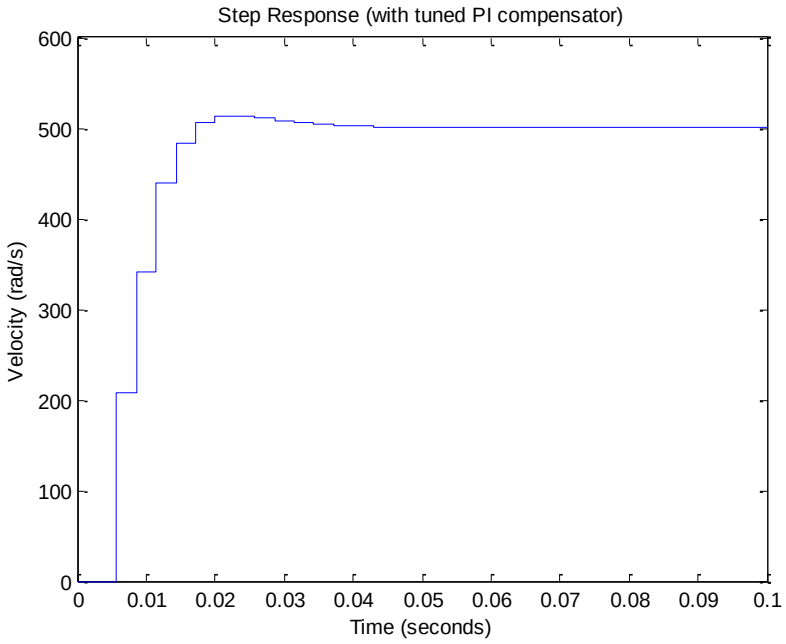


Figure 4-31: Step response (maximum step size) for the PI compensator included control system (simulation)

The following tuned control loop has a transfer function which is illustrated in equation 4-68:

$$H(z) = \frac{\dot{\theta_m}(z)}{E(z)} = \frac{(5.174)z+1.19}{z^2-(4.432)z+1.198} \qquad \text{Equation 4-68}$$

Although the controller should control the position and the pose of the joint, the following transfer function is used for the velocity controller.

4.5.1. Position controller

The position controller in the joints is the outer control loop cascaded to the velocity controller. The control loop is based on a proportional (P) compensator, which is illustrated in figure 4-32.

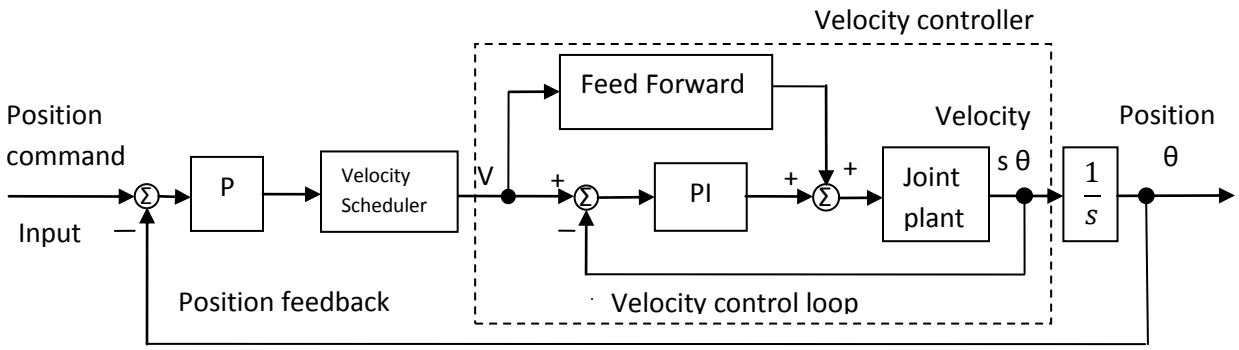


Figure 4-32: Position controller coupled to the velocity controller

The transfer function of the velocity controller block should be converted to the continuous time space (Laplace). The outcome is the velocity controller combined to the joint plant (motor, and gear) transfer function, which gives the velocity of the joint. The velocity controller block is maintained by an integral block (multiplication by s^{-1} in the Laplace space). Since the integral over time of the velocity is the position.

The resulted transfer function should be converted to the discrete time space using the control loop time period. The value for the controller used in Archie is $T_s = \frac{1}{80Hz} = 0.0125ms$.

The position controller loop uses a proportional compensator that drives a velocity scheduler. Since one of the zeros is located outside of the unique circle. The Value of the K_P has an utmost (in the following system the value 131 is the border for instability). By passing the K_P utmost the control loop will be unstable.

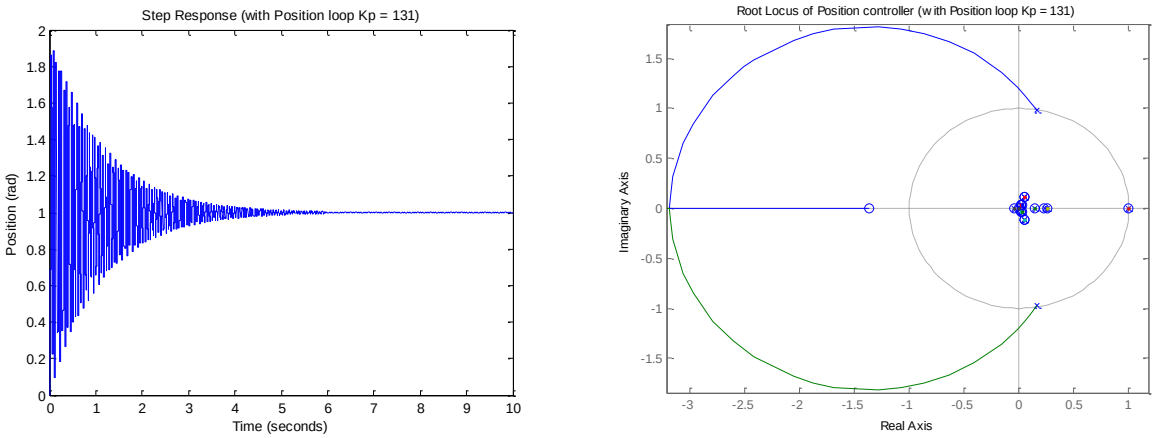


Figure 4-33: Step response and root locus diagram of utmost value for K_P of the position controller (simulation)

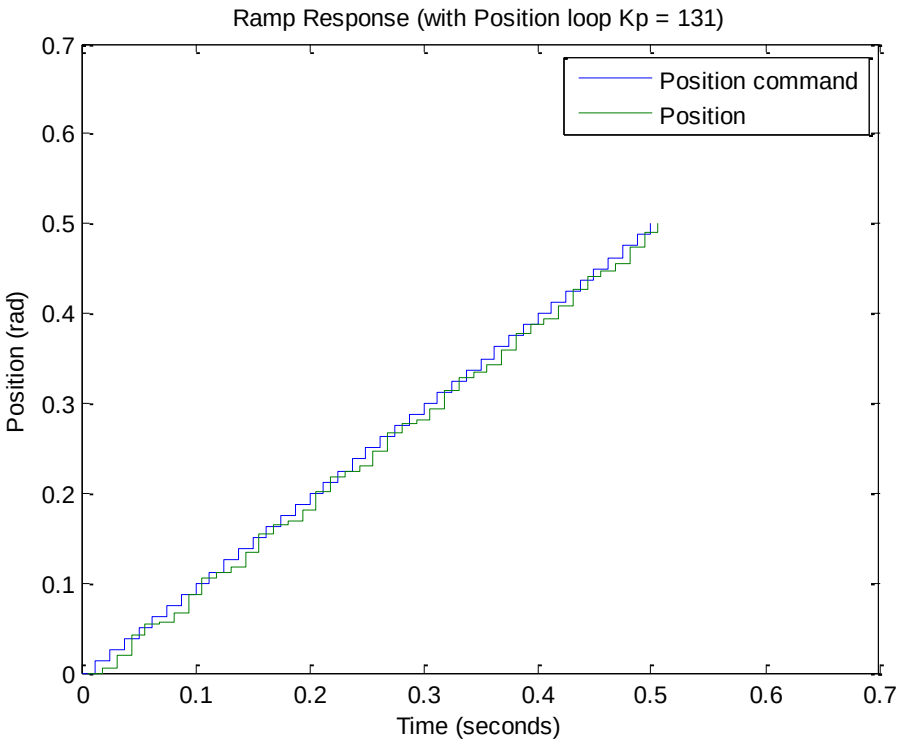


Figure 4-34: Ramp response of the position controller (with K_P = 131) in simulation

Using a low value decreases the response time of the system, and causes steady state error, which is shown in figure 4-35.

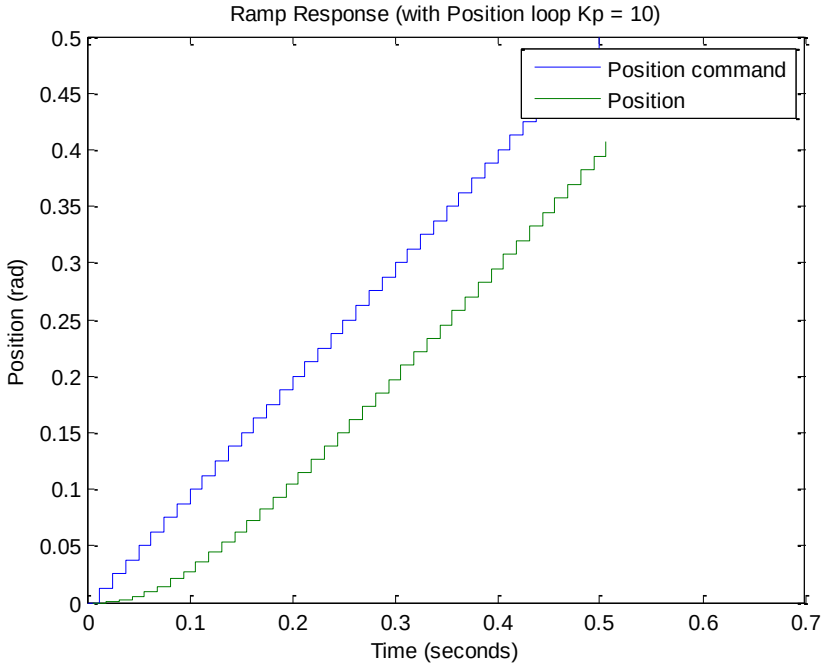


Figure 4-35: Ramp response of the position controller (with K_P = 10) in simulation

In both simulation and the real joint the K_P= 56 is chosen.

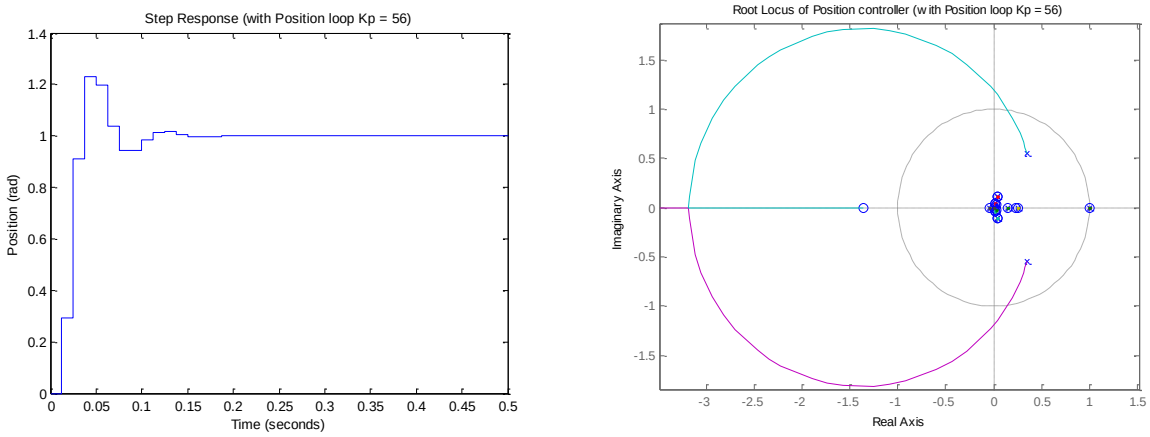


Figure 4-36: Step response and root locus diagram of chosen value for K_P of the position controller (simulation)

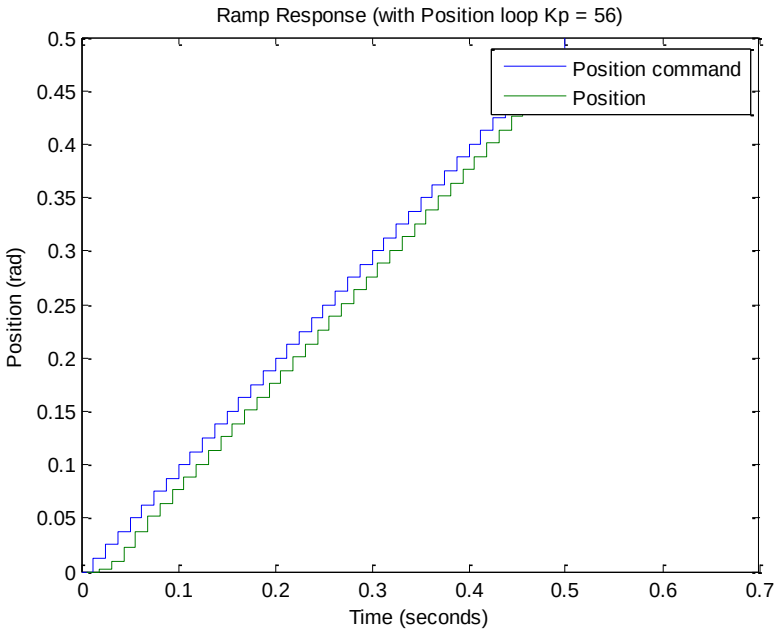


Figure 4-37: Ramp response of the position controller (with K_P = 56) in simulation

The velocity scheduler described before is used to move the joint with certain velocity to the desired position. This scheduling consists of acceleration, declaration and a maximum speed to synchronize the motion of the joints in combinational movements of the robot.

The read joint controller is tested with the following K_P to verify the similarity between the simulation model and the real joint controller. Figure 4-38 shows the ramp response for the real controller.

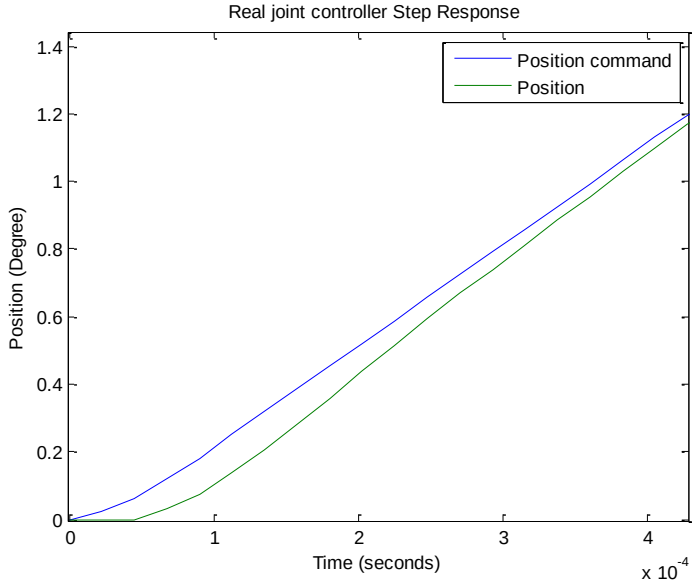


Figure 4-38: Ramp response of the real joint position controller (with $K_P = 56$)

Figure 4-39 shows the position command in comparison with the output of the position controller. The velocity and current of the motor respectively in figures 4-40 and 4-41 for moving from angle 0 to 30 are shown.

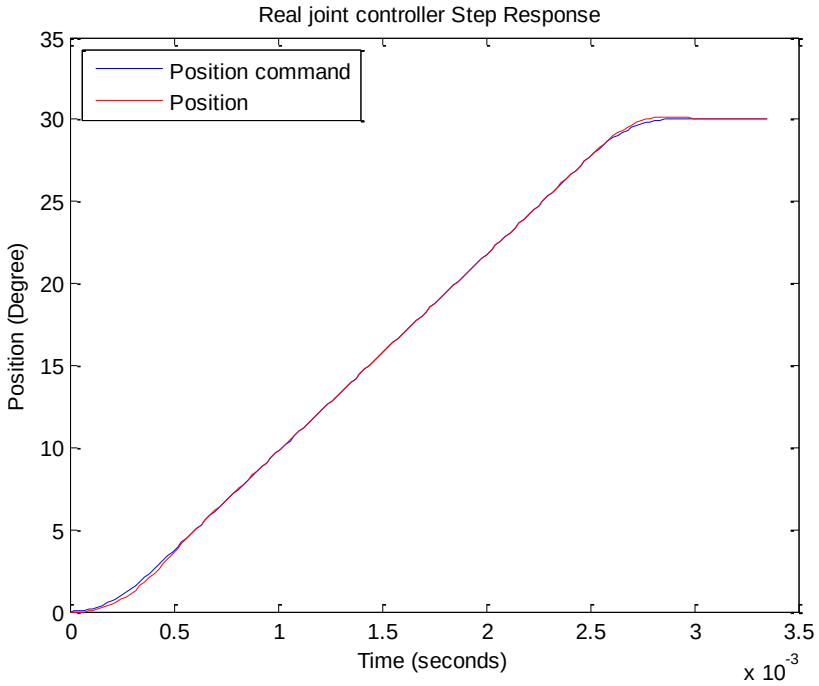


Figure 4-39: Position controller output in compare with position command

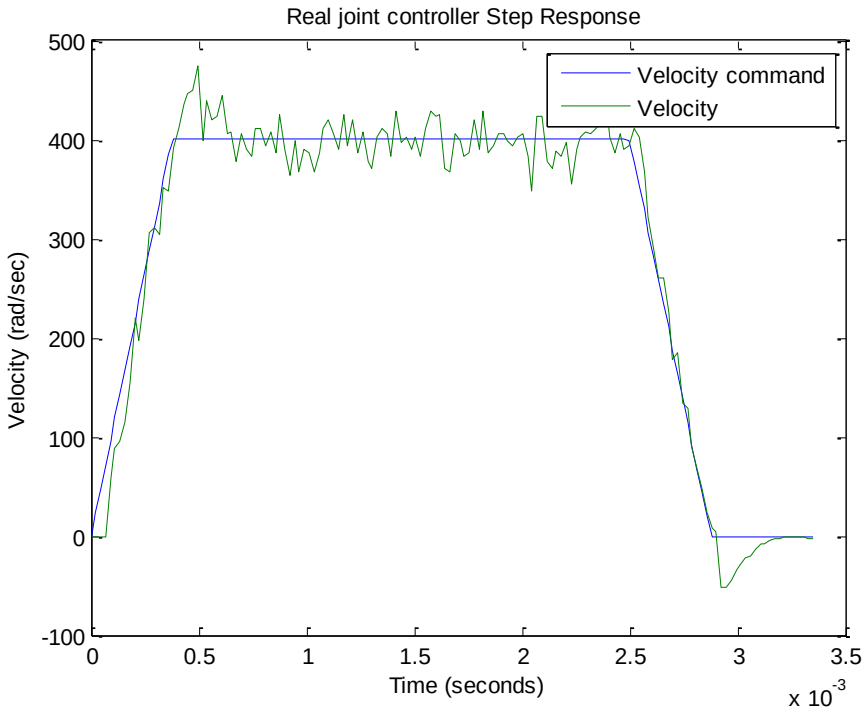


Figure 4-40: Velocity compare with the velocity command in position controller

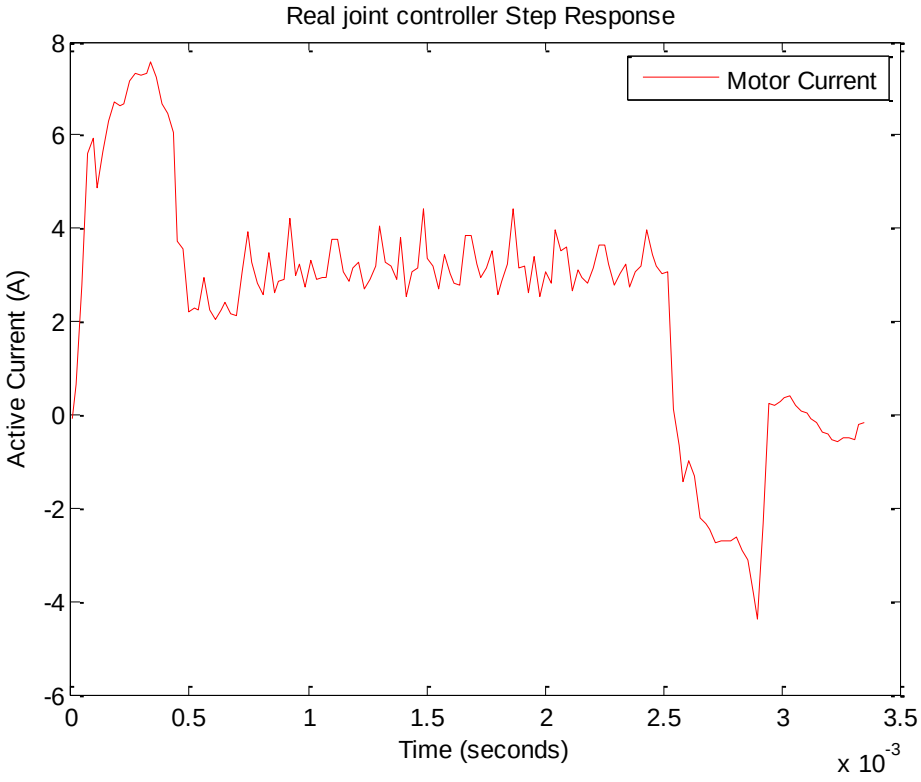


Figure 4-41: Motor current during the joints movement

4.6. Walking sequence

Biped walking algorithms can be distinguished as being either static or dynamic. The distinction is made depending on the location of the centre of mass during motion. For static walking, the centre of mass is always located above a polygon created by external boundaries of the leg base. The biped will remain statically stable if it is paused at any time during its motion. Dynamic walking is generally much faster than static walking. In dynamic walking, inertia effects are considered, and it is possible for the centre of gravity to be outside the supporting area. Human walking patterns are considered to be dynamic (Ian Joseph Marshall, 2002).

4.6.1. Gait analysis

Gait analysis is the study of animal locomotion, including locomotion of humans. Describing human gait requires some specific terms, which are defined in this section.

The gait cycle begins when one foot contacts the ground and ends when that foot contacts the ground again. Thus, each cycle begins at initial contact with a stance phase and proceeds through a swing phase until the cycle ends with the limb's next initial contact. Hence, the human walking step is composed of two different phases:

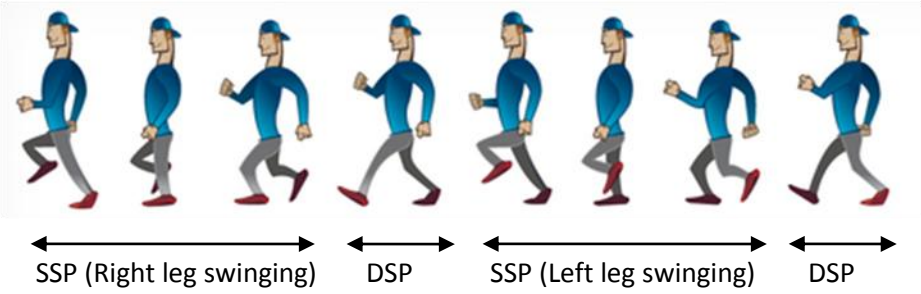


Figure 4-42: Human gait cycle

The first phase is the swing phase or single support phase. This term is used for situations where the body has only one leg on contact surface with the ground. The second phase is called the double support phase; which is used for situations where the body has two isolated contacts surfaces with the ground. In human gait, this situation occurs when the person is supported by both feet. The gait phases of normal dynamic walk consists of eight steps but only

four of them are different as the right and left leg execute the same motion mirrored in the median plane delayed by a half gait cycle.

The following phases are split in more detailed phases. Following is the description for the detailed phases:

1. Release Phase: The infinitesimal period of time when the toes of the rear foot breaks contact with the ground.

2. Single Support Phase (SSP): The phase where only one foot has contact with the ground and the other foot swings. Also the SSP is divided into two phases: one is where the right foot is in contact with the ground (SSP-R) and the other is where the left foot is in contact with the ground (SSP-L).

3. Impact Phase: The infinitesimal period of the time when the heel of one swinging foot strikes the ground.

4. Double Support Phase (DSP): The phase where both feet have contact with the ground. The DSP is divided in two phases: one is where the weight is shifted from the left to right foot (DSP-L) and the other is where the weight is shifted from the right to the left foot (DSP-R).

Thus, human walk can be described as a heel-strike-toe-off gait cycle. The four phases are depicted in Fig. 2.4.3:

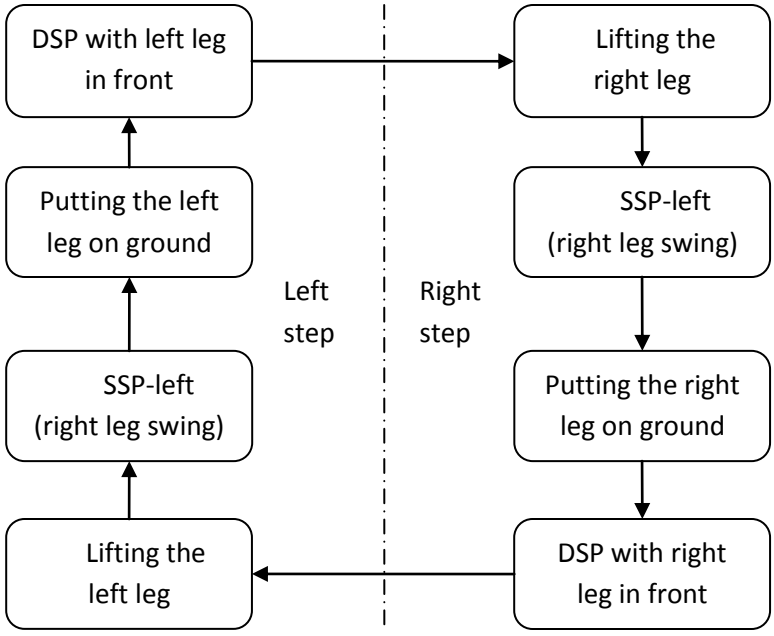


Figure 4-43 : Walking sequence diagram

In the description of the joint controller the system which is described need the values of the feed forward loop such is moment of inertia. The moment of inertia is calculated for a range and it is based on swinging or supporting phase. The feed forward values should be changed based on the phase where the joint is.

4.6.2. Motion planning

As mentioned above, bipedal walking consists of two main phases. Each phase consists of a group of motions. Each motion should be planned for a specific joint of the robot individually. For instance, for the single support phase both the planning of the hip and the planning of the floating leg as well as the supporting leg, allows the robot to move forward.
Figure 4-44 shows a complete semi-cycle walk, which distinguishes the trajectories generated in the lateral plane:

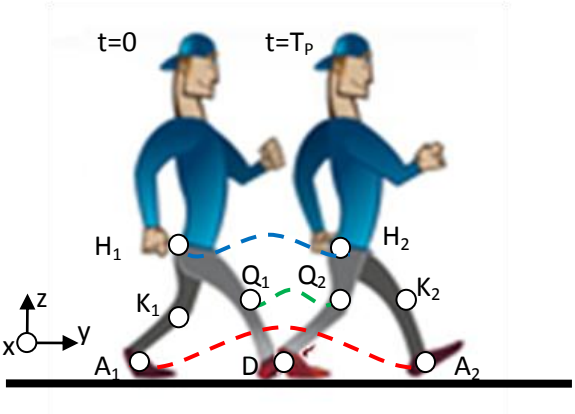


Figure 4-44: Swinging leg gait trajectory (lateral view)

Figure 4-44 outlines the initial state (A_1) and the final state (A_2) of a walking semi-cycle in the lateral plane. In figure 4-45 the simplified model (only lower body) is shown.

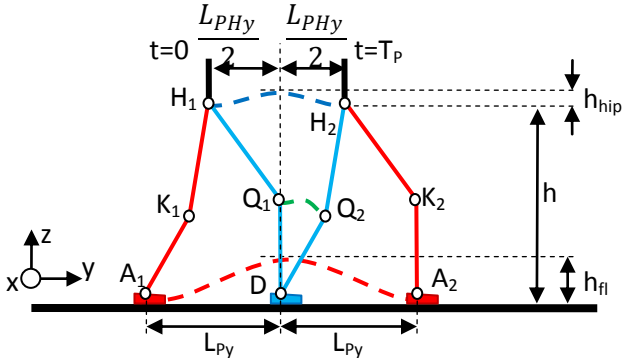


Figure 4-45: Simplified model of swinging leg trajectory (lateral view)

Where:

T_P: time for a single step

L_P: length of the step

L_{PH}: length of hip movement

h: height of the hip

h_{hip}: utmost height of the hip twisting

h_f: utmost height of the gait

In figure 4-45 the trajectory of the hip in the lateral plane is depicted by the blue curve while the trajectory of the swinging leg is shown by the red curve. In the semi-cycle shown in figure 4-45, the fixed point of reference is the point D (the planning is related to this point which is the supporting leg's ankle joint). For the next semi-cycle, the reference point is the point A_2 in respect with the change of the swinging leg to the supporting leg. As the robot moves the reference point changes.

In order to save the balance of the robot, the motion is applied to the frontal joints of the robot as well. Figure 4-46 illustrates the frontal view for the single support phase.

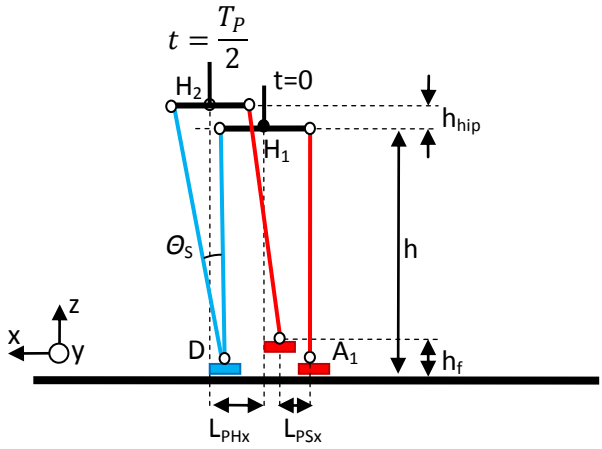


Figure 4-46: Simplified model of swinging leg trajectory (frontal view)

The movement of the joints should conclude a trajectory of the movement for the end effector (the swinging ankle) which provides some desired properties for the robot. Some of these properties are described in following:

- Obstacle passing: Providing certain height and length for the swinging leg trajectory
- Smoothness: Less vibration in the robot during running the gait imitation.
- Stability: Less deviation from the balance poses of the robot.

Providing the following property is related to choosing an appropriate trajectory for the movement of the robot.

4.6.3. Obstacle passing

The obstacles are supposed to have a certain height and length. The trajectory planner decides the appropriate trajectory to cross the obstacles without bumping them. The robot has an utmost for the height and length of the gait. The following restrictions are derived from the direct kinematics and the joints domain restrictions.
Figure 4-47 shows the obstacle with the maximum height, on passing by the swinging leg of the robot.

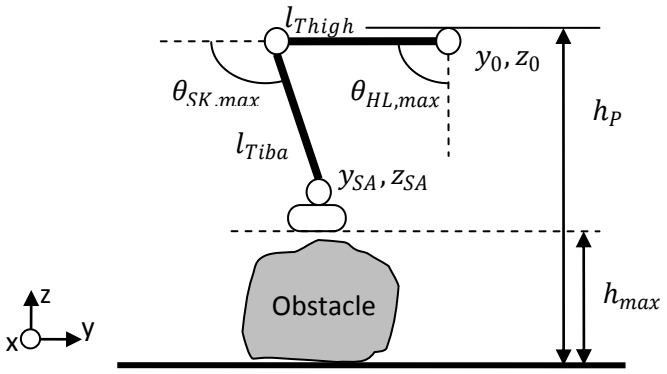


Figure 4-47: Utmost of the gait height

The maximum height that the swinging leg can take is calculated using equation 4-69:

$$h_{max} = h_P - l_{Thigh}.\cos(\theta_{HL,max}) - l_{Tiba}.\cos(\theta_{SK,max} - \theta_{HL,max}) \qquad \text{Equation 4-69}$$

Where:
h_P : is the height of the hip
h_{max} : is the utmost of the height for the swinging leg trajectory in the lateral plane.
$\theta_{HL,max}, \theta_{SK,max}$: are the maximum angles of twisting for the lateral hip and knee joints respectively

The length of the gait is restricted by a limit, which is related to the length of the limbs (lower body) of the robot. Figure 4-48 shows the trajectory of the swinging leg on crossing the obstacle in the lateral plane.

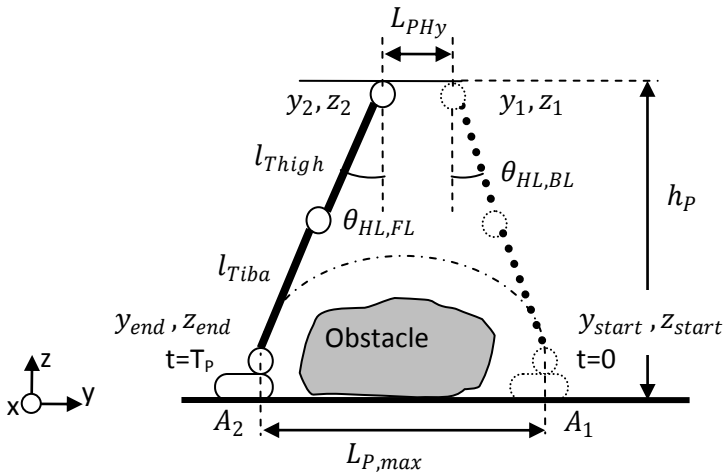


Figure 4-48: Utmost of the gait length

The maximum length that can be taken is calculated using the equation 4-70:

$$L_{P,max} = L_{PHy} + 2.\sqrt{\left(l_{Thigh} + l_{Tiba}\right)^2 - {h_P}^2} \qquad \text{Equation 4-70}$$

Where:

L_{PHy} : is the horizontal movement of the hip in the lateral plane during one step.

Regarding to the parameters of Archie the maximum height and length of the gait are calculated as equations 4-71 and 4-72:

$$h_{max} = 54 - 26.\cos\left(\frac{\pi}{2}\right) - 30.\cos\left(\frac{4.\pi}{6} - \frac{\pi}{2}\right) = 28cm \qquad \text{Equation 4-71}$$

$$L_{P,max} = 19 + 2.\sqrt{(26+30)^2 - 54^2} = 33.83\text{cm} \qquad \text{Equation 4-72}$$

The following calculations are used for Archie's trajectory planner, as the maximum height and length of obstacle that is passable for the robot.

4.6.4. Gait trajectory planning

There are different variants for planning the trajectory of the swinging leg of a humanoid robot for gait imitation. In this thesis, two of them (elliptical and trapezoidal) are studied and

implemented on the robot. Figure 4-49 shows these two trajectories with the related parameters.

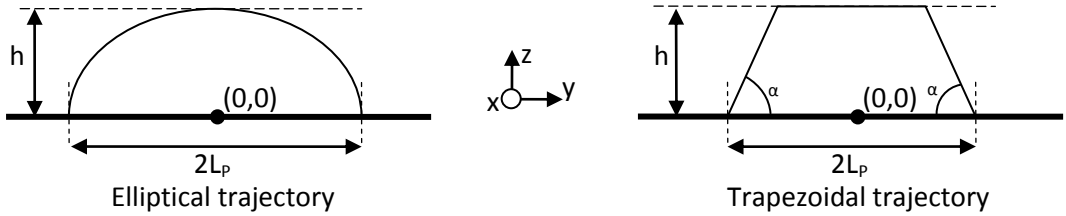


Figure 4-49: Gait's ankle trajectory in lateral plan

Each trajectory has some properties which are:

h: Maximum height

2Lp: Gait length

α: Angle of climbing (as well as descending)

The Elliptical trajectory can be generated using the equations 4-73 and 4-74:

$$y(t) = L_P.\cos\left(\frac{t.\pi}{T_P}\right) \qquad \text{Equation 4-73}$$

$$z(t) = h.\sin\left(\frac{t.\pi}{T_P}\right) \qquad \text{Equation 4-74}$$

Where T_P is the gait time when the swinging phase is executed.

The Trapezoidal trajectory can be generated using the equations 4-75 and 4-76:

$$y(t) = t \qquad \text{Equation 4-75}$$

$$z(t) = \begin{cases} t.ArcCot(\alpha) & : t < h.\cos(\alpha) \\ h & : o.w. \\ (T_P - t).ArcCot(\alpha) & : t > T_P - h.\cos(\alpha) \end{cases} \qquad \text{Equation 4-76}$$

Figure 4-50 and 4-51 shows the generated trajectory of Elliptical, and Trapezoidal form respectively, with h=10, Lp=15 and α=30°.

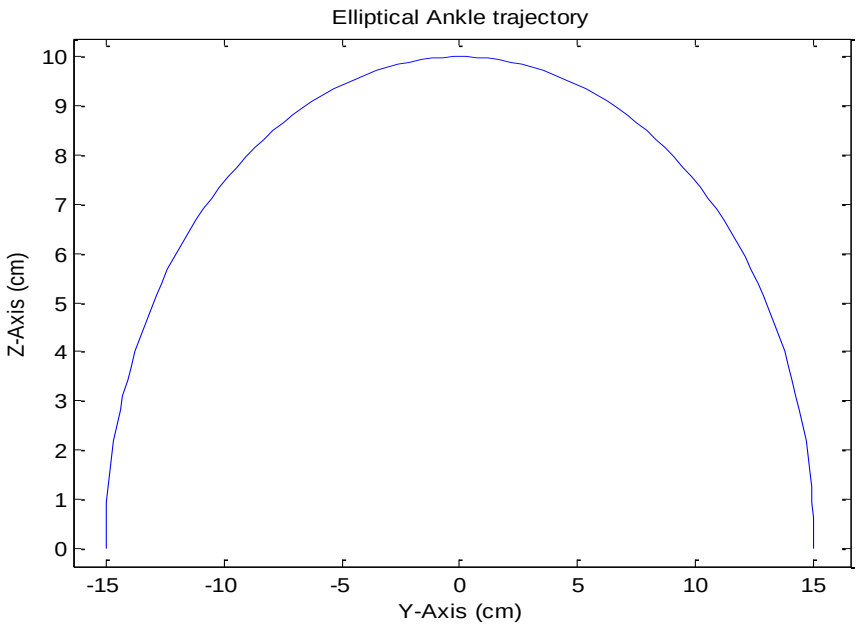


Figure 4-50: Elliptical trajectory

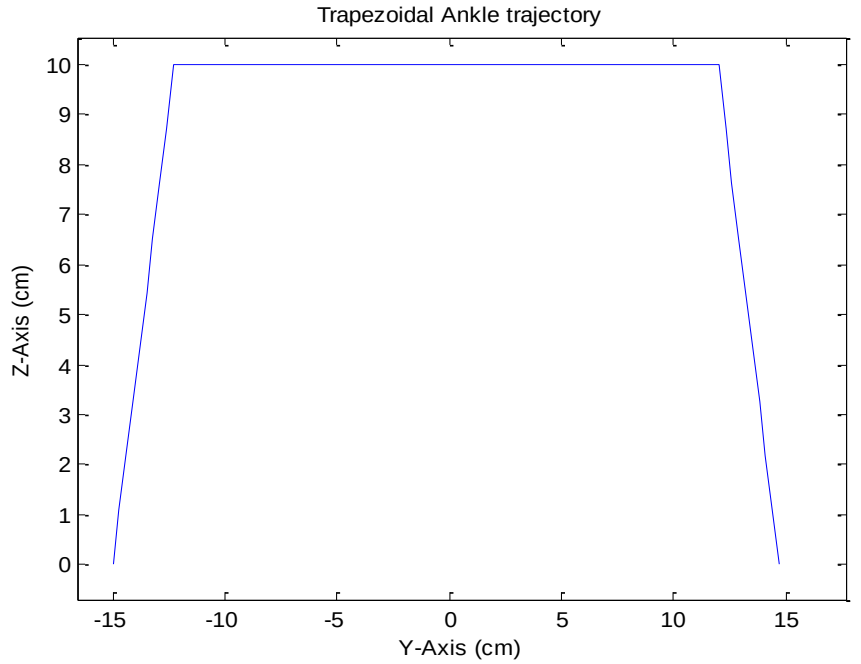


Figure 4-51: Trapezoidal trajectory

The following trajectory should be traversed using the swinging ankle of the robot. For this task five joints are involved. Each joint controller should obey a specific trajectory (in time). The trajectory for each joint is calculated using the inverse kinematics model of the robot.

4.6.5. Inverse kinematics

The goal of inverse kinematics is to compute the vector of the joints of the robot that will cause the end effector to reach the desired point. The following operation is used to find the vectors to approach the end effector to the desired point. To move the end effector on a desired trajectory, the following operation (inverse kinematics) should be applied on the constructive points of the trajectory consequentially. The result of the following process is the trajectory for the joints on the limb.

The robot is controlled indirectly, from the movement of the joints.

Since direct kinematics is:

$$X = f(\boldsymbol{\Theta}) \qquad \text{Equation 4-77}$$

Where:

X: is the vector of position of the end effector.

Θ: is the vector of joints (degrees of freedom) of the kinematics chain.

Inverse kinematics, determines the configuration that a robot must take to obtain a certain position and orientation of the end effector.

$$\Theta = f^{-1}(X) \qquad \text{Equation 4-78}$$

The inverse kinematics problem can be approached from many points of view; however, a few are applicable to this case. Usually solving the inverse kinematics problem for the joints of a robot using analytical methods is not possible, because the analytical methods cannot mathematically solve an exact solution by directly inverting the forward kinematics equations, which is only possible on relatively simple kinematics chains. Numerical methods use approximation and iteration to converge on a solution. However, they tend to be more expensive in processing load, but they have far more general purpose to use (Steve Rotenberg, 2005). One of the popular methods is the gradient descent. The gradient descent is based on finding the values of a function for a certain point by knowing the value of the function and the value of the derivative of the function for a near point.

$$\frac{\Delta f}{\Delta x} \approx \frac{df}{dx} \qquad \text{Equation 4-79}$$

$$\Delta f \approx \Delta x . \frac{df}{dx} \qquad \text{Equation 4-80}$$

$$f(x + \Delta x) \approx f(x) + \Delta x . \frac{df}{dx} \qquad \text{Equation 4-81}$$

To extend the following methods to the vectors, the Jacobian matrix is used. The Jacobian matrix contains all of the information necessary to relate a change in any component of Θ to change in any component of X.

$$J(\Theta) = \left(\frac{\partial X_i}{\partial \theta_j}\right)_{i,j} \qquad \text{Equation 4-82}$$

Since the gradient descent formula can be extended to the vector form using the Jacobian matrix.

$$\Delta X \approx J(\Theta).\Delta\Theta \qquad \text{Equation 4-83}$$

That the X and Θ vectors are:

$$X = \begin{bmatrix} y_{SA} \\ z_{SA} \end{bmatrix} \quad , \quad \Theta = \begin{bmatrix} \theta_{HL} \\ \theta_{SK} \end{bmatrix} \qquad \text{Equation 4-84}$$

Where:

y_{SA} , z_{SA} : are the positions of the swinging ankle in the lateral plane respectively.

θ_{HL} , θ_{SK}: are the angle of the swinging hip and the swinging knee respectively.

For instance the inverse kinematics problem is applied to the swinging leg of the robot in order to find the trajectory of the lateral hip and the knee joint for moving the ankle on desired trajectory. Figure 4-52 shows the swinging leg moving on a desired trajectory.

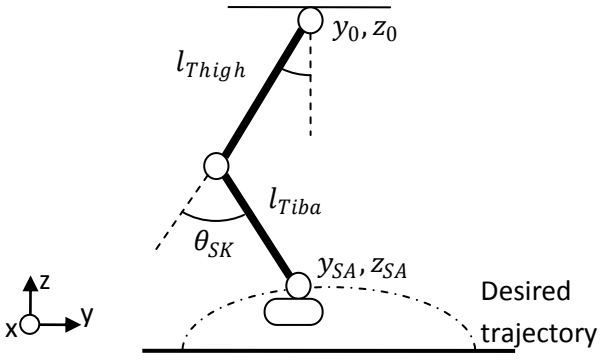


Figure 4-52: Swinging leg and the desired trajectory

From the direct kinematics chain the position of the end effector is resulted from the equations 4-85 and 4-86:

$$y_{SA} = y_0 + l_{Thigh}.\sin(\theta_{HL}) - l_{Tiba}.\sin(\theta_{SK} - \theta_{HL}) \qquad \text{Equation 4-85}$$

$$z_{SA} = z_0 - l_{Thigh}.\sin(\theta_{HL}) - l_{Tiba}.\sin(\theta_{SK} - \theta_{HL}) \qquad \text{Equation 4-86}$$

Since the Jacobian matrix $J(\Theta)$ for this problem is:

$$J(\Theta)=\begin{bmatrix}\frac{\partial y_{SA}}{\partial \theta_{HL}} & \frac{\partial y_{SA}}{\partial \theta_{SK}}\\ \frac{\partial z_{SA}}{\partial \theta_{HL}} & \frac{\partial z_{SA}}{\partial \theta_{SK}}\end{bmatrix}=\begin{bmatrix}l_{Thigh}.\cos(\theta_{HL})+l_{Tiba}.\cos(\theta_{HL}-\theta_{SK}) & -l_{Tiba}.\cos(\theta_{SK}-\theta_{HL})\\ l_{Thigh}.\sin(\theta_{HL})+l_{Tiba}.\sin(\theta_{HL}-\theta_{SK}) & l_{Tiba}.\sin(\theta_{SK}-\theta_{HL})\end{bmatrix}$$

Equation 4-87

The $\Delta\boldsymbol{\Theta}$ vector is calculated from the equation 4-88.

$$\Delta\Theta \approx J^{-1}(\Theta).\Delta X$$ Equation 4-88

Where the $J^{-1}(\Theta)$ is the inverse of the Jacobian matrix and is found using the equation 4-89.

$$J^{-1}(\Theta)=\frac{1}{\left(\frac{\partial y_{SA}}{\partial \theta_{HL}}.\frac{\partial z_{SA}}{\partial \theta_{SK}}\right)-\left(\frac{\partial z_{SA}}{\partial \theta_{HL}}.\frac{\partial y_{SA}}{\partial \theta_{SK}}\right)}\begin{bmatrix}\frac{\partial z_{SA}}{\partial \theta_{SK}} & -\frac{\partial y_{SA}}{\partial \theta_{SK}}\\ -\frac{\partial z_{SA}}{\partial \theta_{HL}} & \frac{\partial y_{SA}}{\partial \theta_{HL}}\end{bmatrix}$$ Equation 4-89

Since the inverse Jacobian matrix consists of the inverse determinant fraction, there are positions which the inverse Jacobian matrix cannot be calculated (causes infinity). Moreover, in the cases that the Jacobian matrix is not square (number of joint DOF is not equal to number of DOF for the end effector) the inverse Jacobian matrix cannot be calculated. To use a more global method the pseudo-inverse is used for inverting the Jacobian matrix (equation 4-90).

$$J^{*}=(J^{T}J)^{-1}J^{T}$$ Equation 4-90

By using the pseudo-inverse method a non-square matrix will be invertible.
For instance, the calculations that are used to find the trajectory of the joints are implemented.

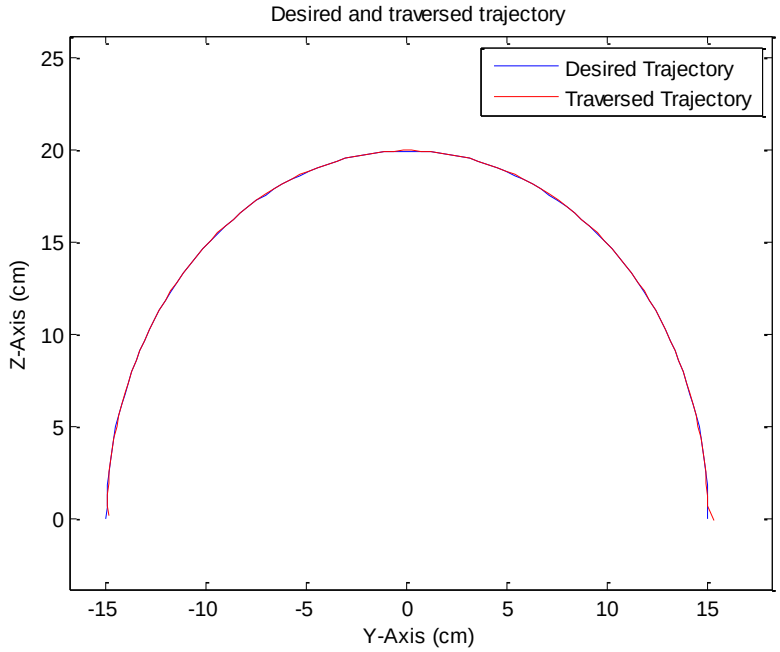


Figure 4-53: Desired trajectory and the traversed trajectory resulted from inverse kinematics for the ankle (simulation)

In figure 4-53 the desired trajectory is shown and compared with the traversed trajectory of the ankle joint. The traversed trajectory is resulted from the inverse kinematics calculations for the swinging leg. The trajectory which should be traversed by the lateral hip joint and the knee joint in order to result the desired ankle trajectory are shown in figure 4-54 and figure 4-55.

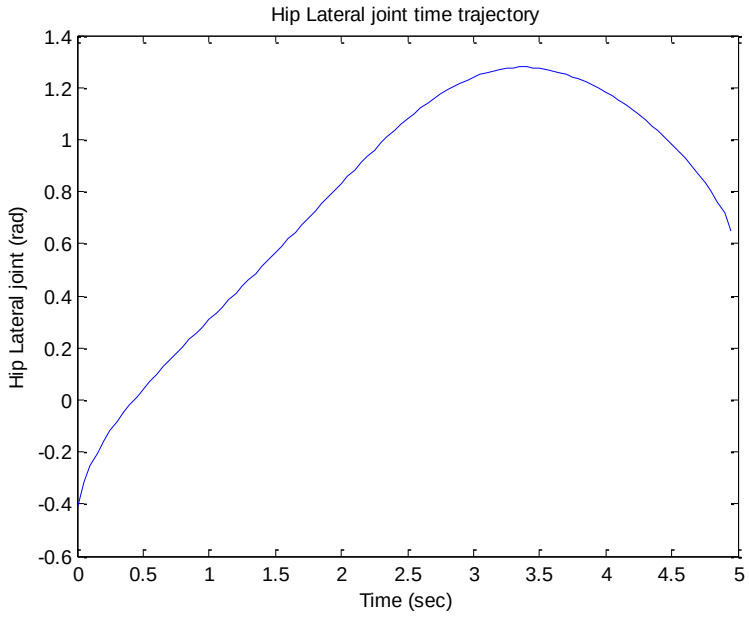


Figure 4-54: Hip lateral joint desired trajectory on time

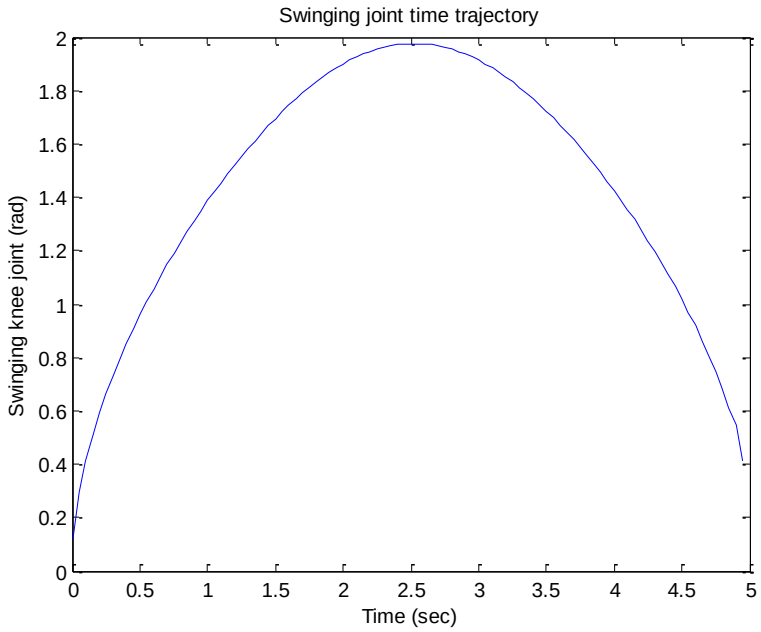


Figure 4-55: Knee joint desired trajectory on time

The trajectory error resulted from the movement of the swinging leg's ankle is illustrated in figure 4-56.

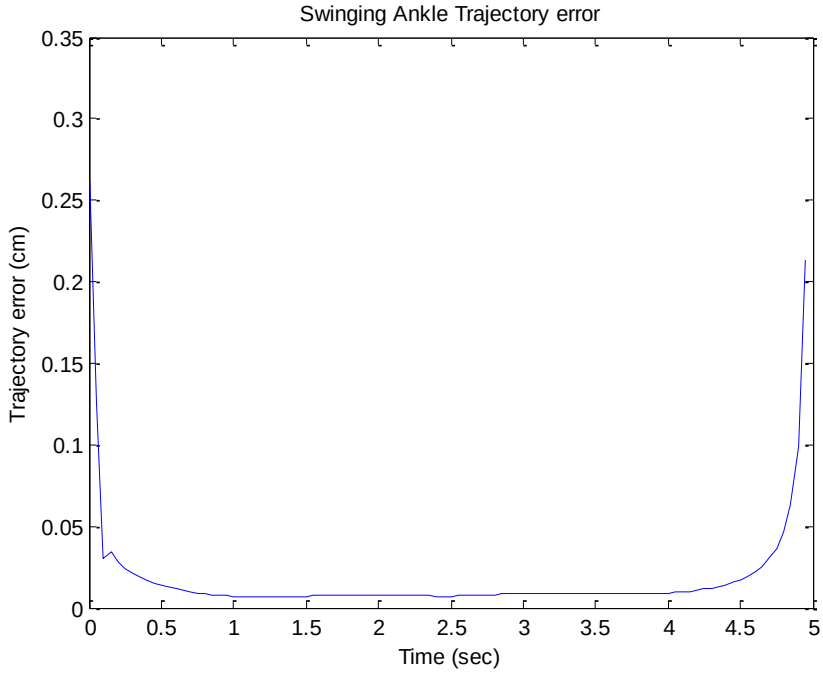


Figure 4-56: Trajectory error of the swinging leg's ankle (simulation)

Figure 4-56 depicts the magnitude of the error which is less than 0.25 cm (in worst case) and is acceptable for the robot.

4.6.6. Pre-calculated inverse kinematics

In this method, using the inverse kinematics of the robot's leg, the pose of the lateral hip joint and the knee joint is calculated to move the end effector (the ankle joint) on all its possible points. By using the following pre-calculations, the robot could move the end effector without calculating the inverse kinematics during the normal operation. The outcomes of this method are 2D arrays per each joint (i.e., for the lateral hip joint one 2D array and for the knee another 2D array). These 2D arrays are storing the angle values of the joint based on the desired end effector position (from the direct kinematics calculation). In addition, by using this method the prohibited values of the joints are restricted. Moreover, the movement of the end effector caused by the motion of the joints is more predictable than using the inverse kinematics directly on the robot. Figures 4-57 and 5-58 are the 3D illustrations of the lateral hip and knee joint respectively.

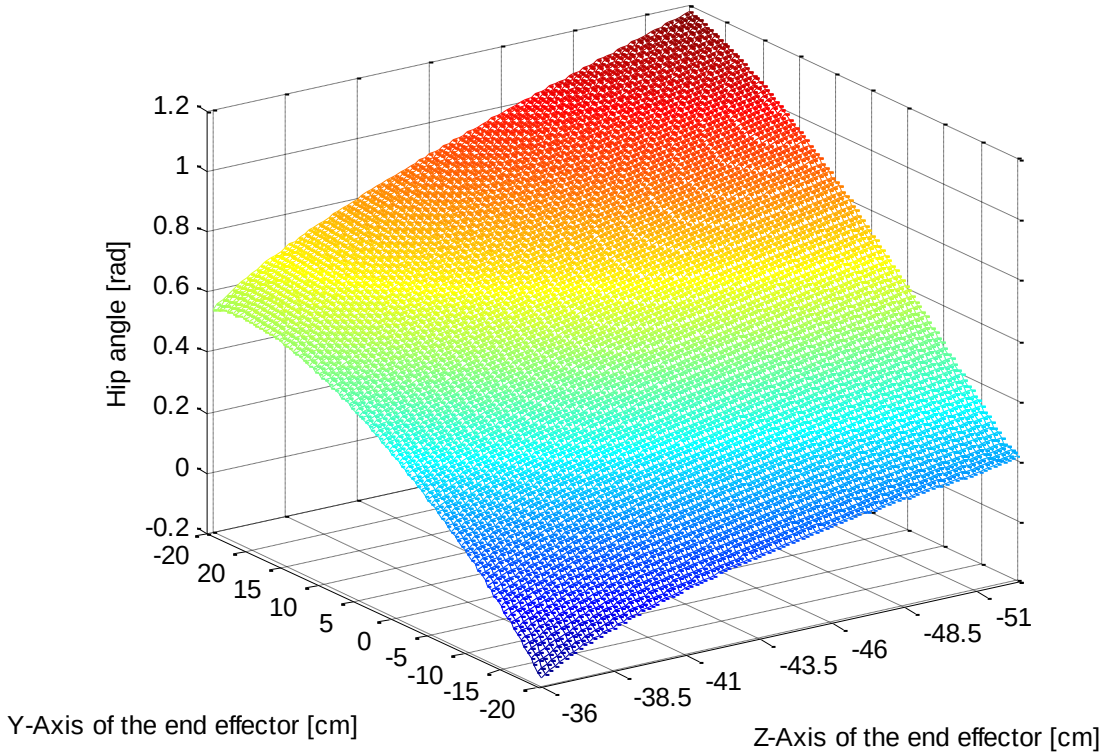


Figure 4-57: Pre-calculated inverse kinematics of the lateral hip joint

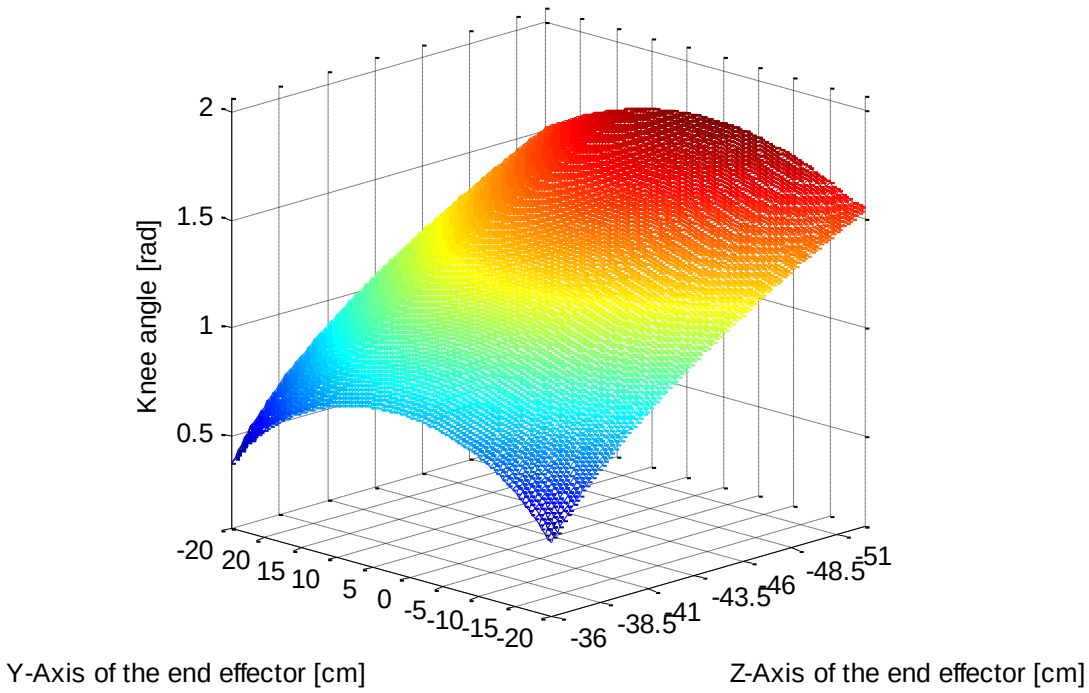


Figure 4-58: Pre-calculated inverse kinematics of the knee joint

Using the pre-calculated inverse kinematics reduces the processing load on the central controller processor; however it requires higher memory usage to store the pre-calculated inverse kinematics values for each joint of the robot (10Kbyte memory per joint).

Chapter 5

5. Realization and Implementation

5.1. Introduction

In this chapter the implementation of the control system designed for Archie is presented. The control system in Archie is designed on the distributed computer architecture basis. A central controller sends commands to the individual controllers through a data network. Each individual controller is intended to control a single joint in the robot and is connected to the central controller. Controlling a joint includes controlling the position and velocity.

Communication process management is time consuming, and it imposes a heavy load on the central processor. Therefore, in this thesis, the communication process management is implemented on a Fundamental Programmable Gate Array (FPGA). By using this approach, the power consumption in the system can be reduced significantly and as a result, the central processor can run other tasks in acceptable times.

There are three type of motors used in the joints of Archie (i.e., RC servo, Brushed DC and Brushless DC motor). The types of the motors have been selected, based on the joint and the torque. Consequentially, the design of the controllers is based on the motor type.

5.2. Joints with RC servo motors

In Archie's upper body (i.e., head, neck and hands) miniature servo motors, which contain built-in controllers (i.e., Dynamixel, RX-64), have been used. A servo motor is shown in figure 5-1. Although it is small, a servo motor's performance is remarkable.

Figure 5-1: RX-64 servo motor used in Archie

Specification	Value	unit
Weight	125	g
Dimension	40x61x41	mm
Gear ratio	1:200	
Supply voltage	18	v
Current (Full load)	1200	mA
Output torque	6	Nm
Speed (No load)	490	Deg/s
Communication speed	1	Mbps
Protocol	RS485 (8bit+1stp)	

Table 5-1: Specification table of RX-64

The RX-64's controller uses EIA-485 communication bus. The central controller sends appropriate data to the motors through a specified hardware based data sequencer. This hardware is integrated to the Data Acquisition Unit (DAU).

Each motor controller uses a specific number (i.e., ID) over the shared bus (i.e., data network) to distinguish itself from the other motor controllers. The protocol used by these motors is shown in figure 5-3.

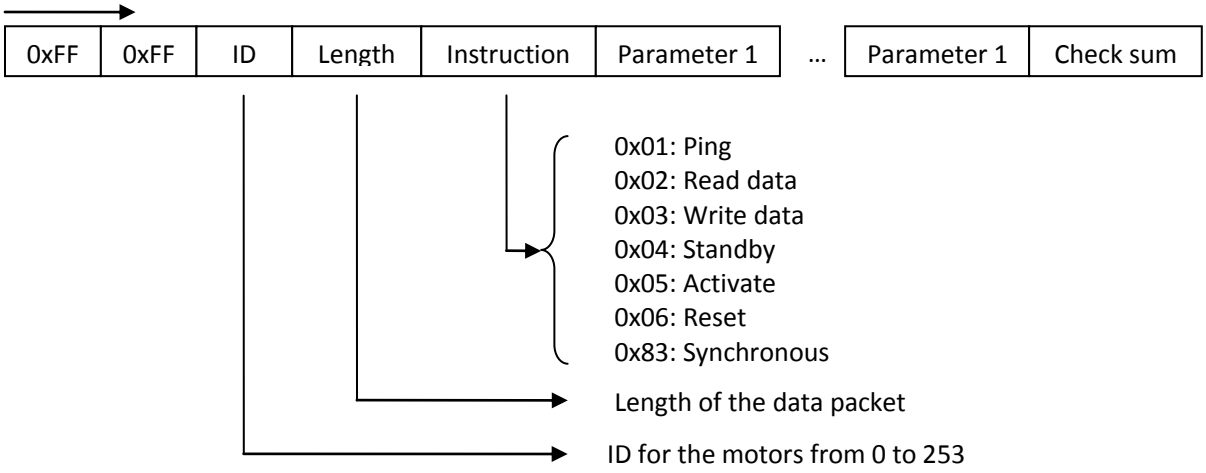


Figure 5-2: Servo motors command packet structure

5.3. Brushed DC motor based joints

Archie's Toes, Ankle (i.e., frontal movement of the Ankle) and Hip (i.e., transversal movement of the hip) are based on brushed DC motors (i.e., Faulhaber, 2342 series). The main reason for choosing brushed DC motors in these joints is the mechanical space restrictions. The motors can provide 16mNm torque on the output shaft without gear. Using a harmonic drive (Type of Gear head) with the ratio of 1:100, 1.6Nm the maximum torque can be provided on the joints.

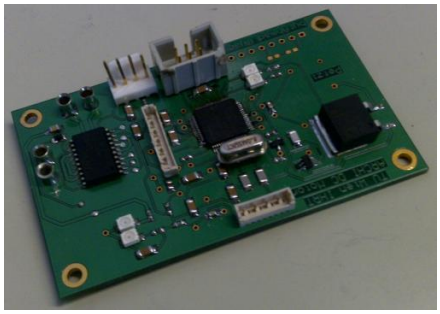


Figure 5-3: DC motor controller

To control the DC motors a velocity controller is used that runs cascaded to a position controller. The following control loop (shown in chapter 4 in figure 4-13) is based on the embedded controller XC164 (Infineon, 2001) which runs at 40MHz. The power stage is based on the ATA6824 (Atmel, 2008) which uses four external power MOSFET (i.e., SUD50N04, Vishay, 2006) for driving the motor.

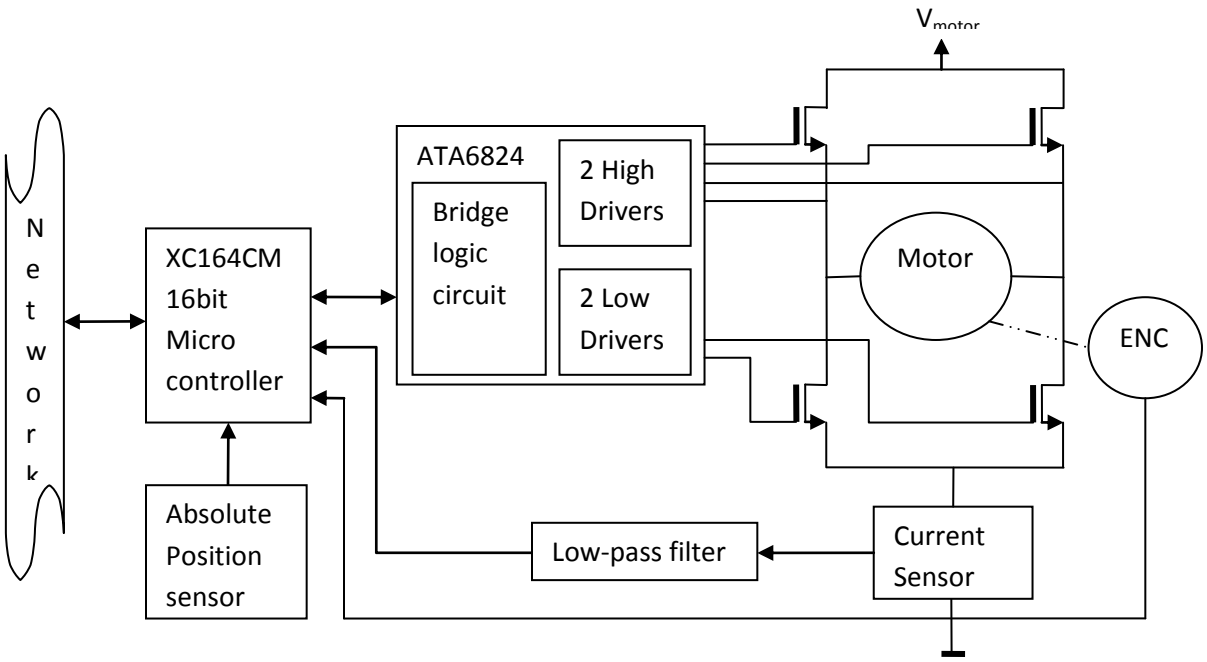


Figure 5-4: DC motor controller hardware

The current sensor is based on the current shunt monitor chip (INA139, Texas Instruments, 2001). The sensor measures the current passing through the shunt resistor. The current passing through the shunt resistor comes from the motor and is affected with noise and Pulse Width Modulation (PWM), and therefore, it needs to be filtered for the measurement. The filter output is measured using an internal digital to analog converter in the microcontroller that is used to controlling the torque loop.

In the DC motor controller, the torque control loop is used to protect the system from possible crashes in feasible collisions, which may occur during a normal operation. The velocity control loop moves the joint to the desired position with a constant velocity. This control loop is important to have synchronization on multi-joint-combinational movements. The outer position control loop is used mostly to correct the position of the joint.

5.4. Brushless DC motor based joints

Archie's Knee, Ankle (i.e., lateral movement of the Ankle) and Hip (i.e., lateral and frontal movement of the hip) are made of DC brushless motors (i.e., Maxon-Motors, EC45). The reason

for using DC brushless motors is the necessity of high mechanical torque in these joints. Here, the brushless motors are combined with a harmonic gear (i.e., Harmonic Drives Systems Inc., 20-160-874405-6) to provide more output torque.

In this section, the mechanical construction has been explained.

The motor controller used to control the brushless based joint and the modular motor-harmonic gear combination is shown in figure 5-5.

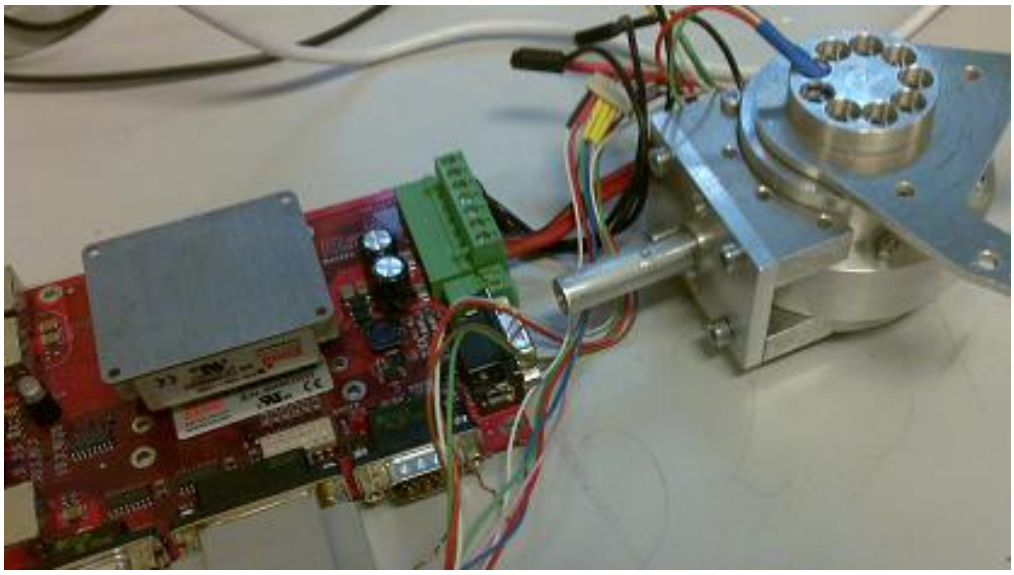


Figure 5-5: Brushless motor controller connected to the joint module

The specifications of the joints have been described in table 5-2.

Parameter	Calculations	Value	Unit
Assigned Power Rating	50(motor power) x 77%(total efficiency)	38.5	Watt
Max. Speed (No load)	6800 (Motor Max. Speed) / 160 (Gear ratio)	42.5	rpm
Stall torque	780 (Motor stall torque) x 160 (Gear ratio)	124.8	Nm
Max. Continuous torque	81.4(Motor cont. Torque) x 160 (Gear ratio)	13.02	Nm

Table 5-2: Output parameters resulted from the constructed joint

5.4.1. Brushless motor advanced controller

The brushless motor based joints use an advanced motion controller (i.e., Whistle, Elmo Motion Control, 2006) as a power stage. The controller block diagram is shown in figure 5-6.

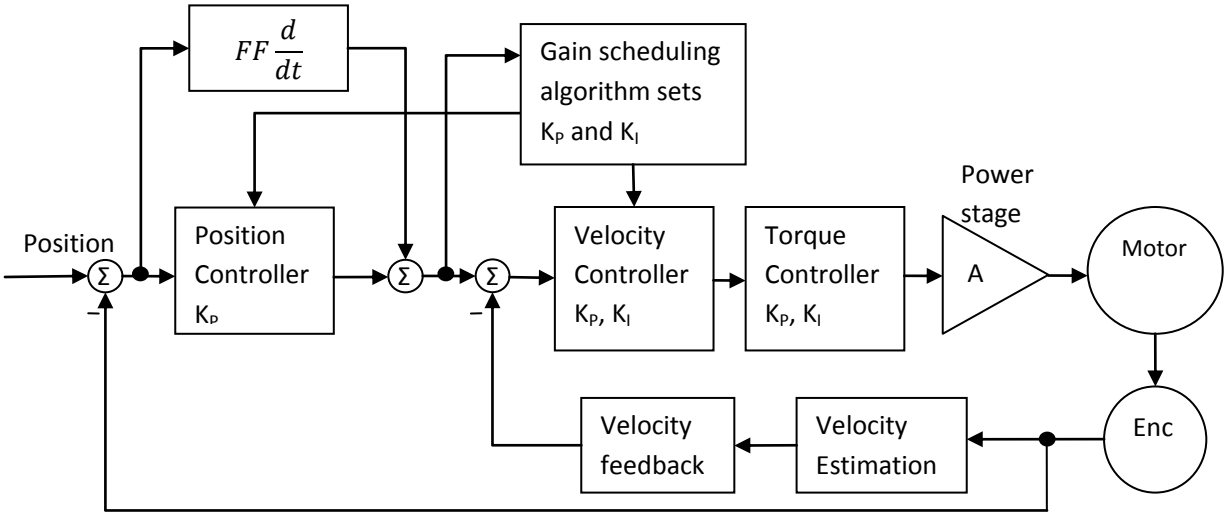


Figure 5-6: Elmo Whistle motion controller

The controller consists of three cascaded controllers to control the position, velocity and the torque of the motor. The torque controller block is illustrated in figure 5-7 (Elmo Motor Controller, 2009).

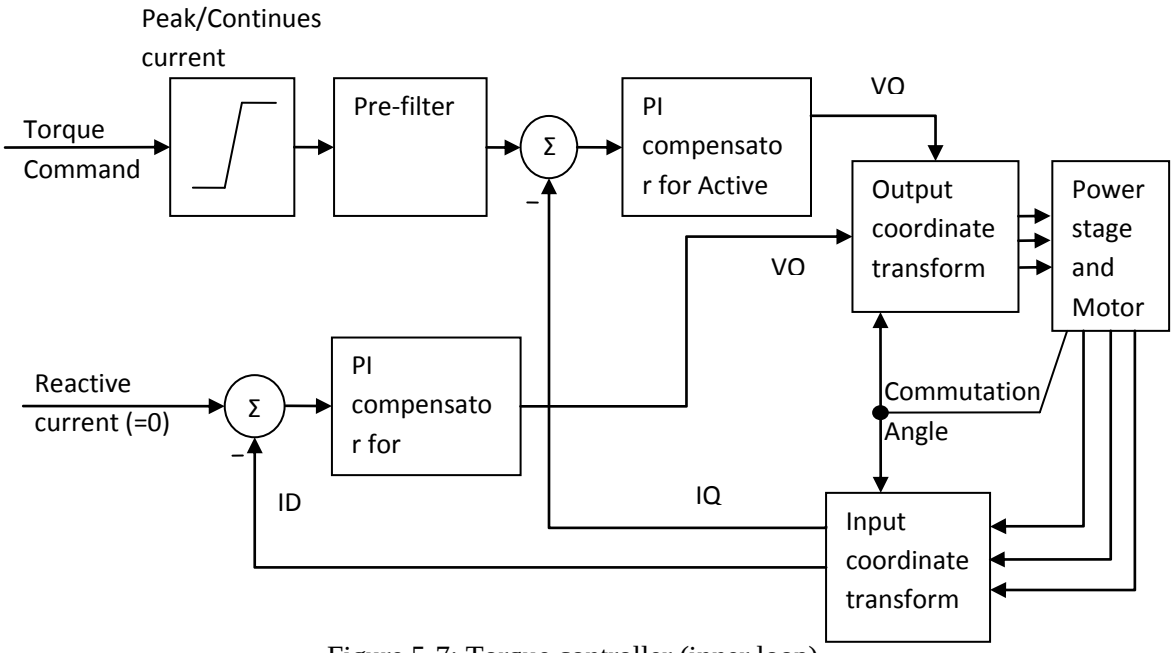


Figure 5-7: Torque controller (inner loop)

To prevent windup in the integral component of the torque control, the following subsystem is added to the system which is shown in figure 5-8 (Elmo Motion Control, 2009).

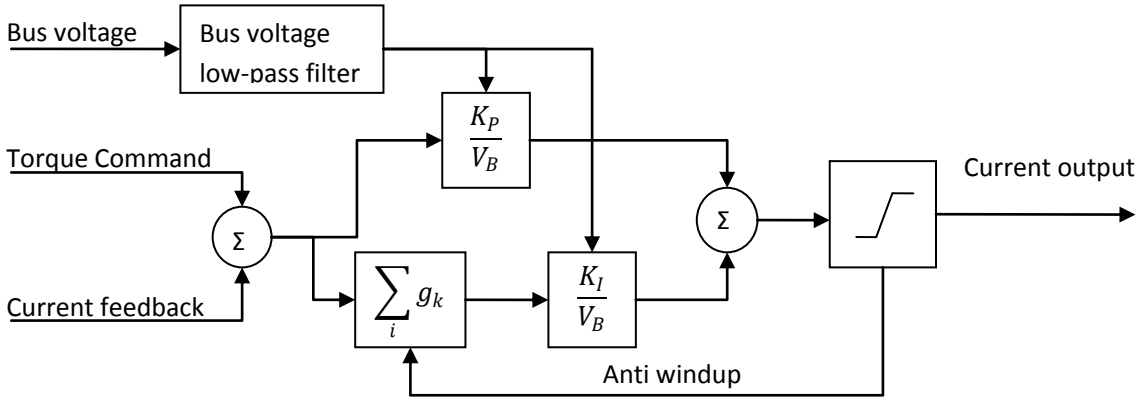


Figure 5-8: Torque controller with anti windup (inner loop)

The velocity controller's block diagram is depicted in Figure 5-9 (Elmo Motion Control, 2009).

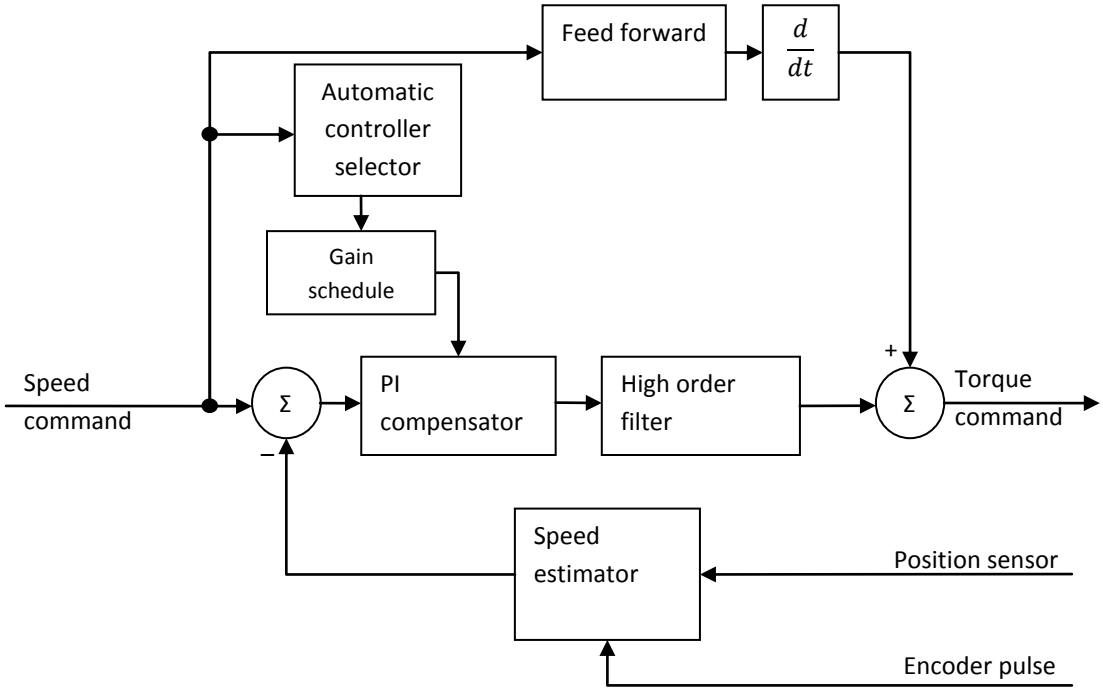


Figure 5-9: Velocity controller used for the burhsless motors

The figure 5-10 shows how the speed controller gets the commands from the position controller (Elmo Motion Control, 2009).

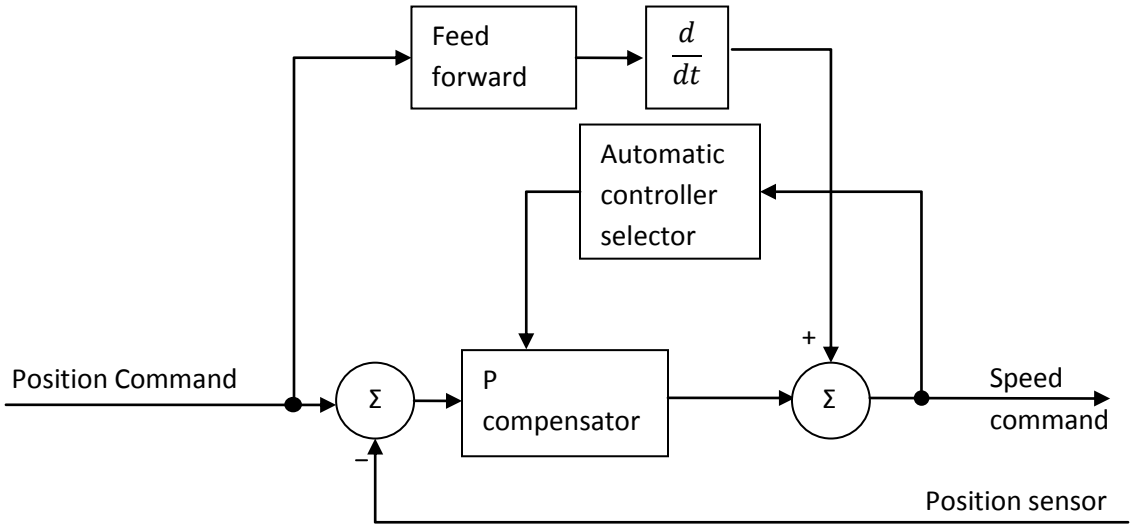


Figure 5-10: Brushless motor position controller (outer loop)

The controller bandwidth is shown in figure 5-11 (Elmo Motion Control, 2009).

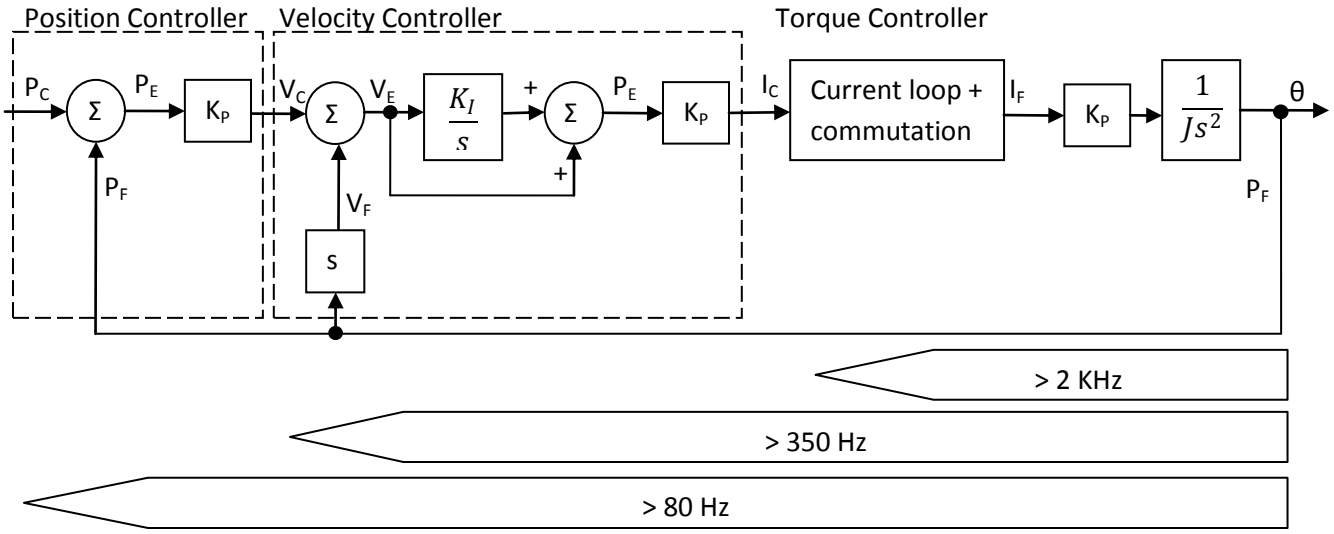


Figure 5-11: Brushless motor's motion controller bandwidth

5.4.2. Absolute positioning

In the brushless based joints, a novel method has been used in this thesis to find the absolute position. The harmonic drive consists of three parts that are shown in figure 5-12. These three parts are wave generator spline, flexible spline and circular spline. The wave generator spline is used as the input to reduce the velocity and to get higher torque in the output. This is coupled to the rotor of the brushless motor. Moreover the flexible spline is the output and it is coupled to the moving link (not to the link that is coupled to the joint frame) and the circular spline is fixed on the frame.

Figure 5-12: Components of the harmonic drive gear

As it is described in the block diagram of the magnetic rotary encoder (i.e., Austrian Micro Electronics, AS5134), different types of outputs are available. A and B outputs are the emulations of the incremental encoder and have been used to giving more accuracy to the brushless controller to generate sinusoidal output for the brushless motor windings. The absolute position output of the chip is used to calculate the absolute position of the joint (i.e., the moving joint).

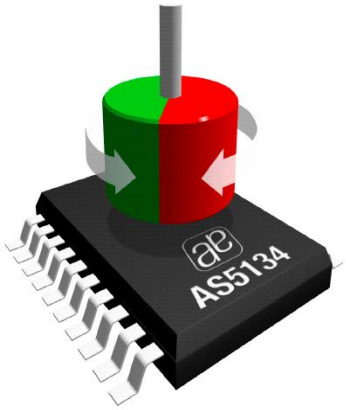


Figure 5-13: AS5134 magnetic rotary encoder with the magnet on the top (Datasheet of AS5134)

The rotor shaft crosses the flexible spline from its center and it is coupled directly to the wave generator spline as well as the rotor. A cylindrical magnet is attached in the end of the shaft which is sensed by the magnetic rotary encoder (shown in figure 5-13). The magnetic rotary encoder is then connected to the output of the harmonic gear that is actually the flexible spline. The result of this configuration will cause drifting by sensing the position of the magnet for each turn in the harmonic gear output.

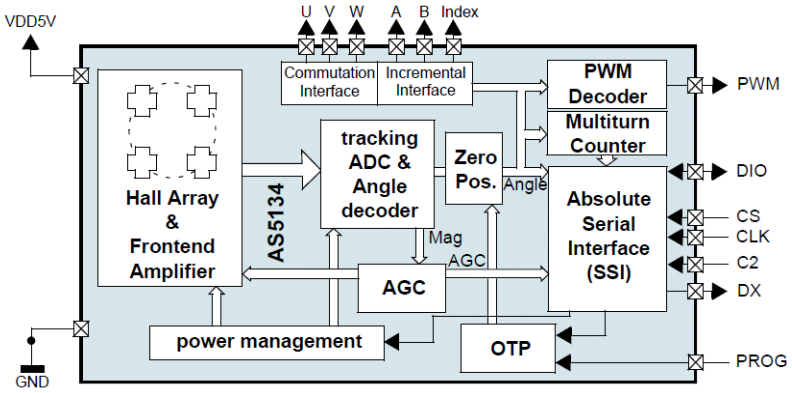


Figure 5-14: Block diagram of AS5134 magnetic rotary encoder (from datasheet of AS5134)

By rotating the rotor shaft in a direction, the output will rotate in the opposite direction with a certain ratio (specified for the harmonic drive, e.g., 1:160). Regarding the described mechanism of placing the encoder on the output and sensing the position of the magnet coupled to the rotor of the brushless motor, there is 362.25^{o} degrees per revolution.

$$\theta = 360° + \frac{360°}{160} = 360° + 2.25° = 362.25° \qquad \text{Equation 5-1}$$

There is a magnet on the rotor generating a pulse in the hall sensor attached to the frame. When a pulse is received, the absolute value of the magnetic rotary encoder is captured. This value will change with each revolution and it is used for calculate the position of the moving link in the joints. Figure 5-15 depicts the magnet in the center and the output of the joint that is coupled to the moving link.

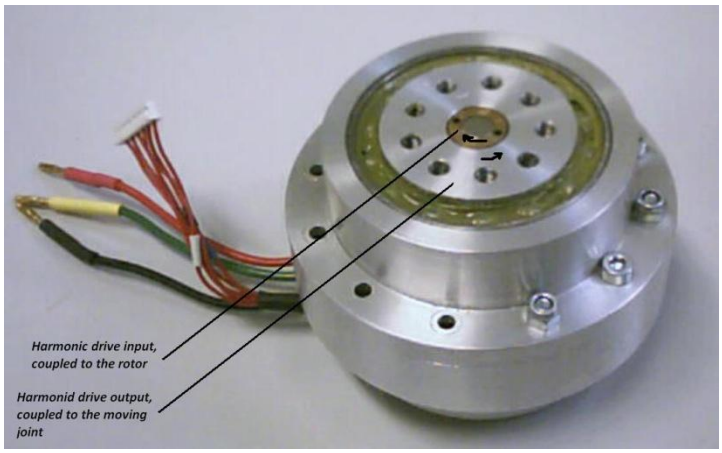


Figure 5-15: The brushless motor based joint and the position of the encoder

To calculate the absolute position of the joint, there is a calibration which relates the resulted value to the real position of the moving link. This calibration is valid until the next change happens in the position of the moving link and the situation of the attachment between the two magnets (one magnet is used for the magnetic rotary encoder and the other is placed on the rotor and generates a pulse in the hall sensor).

Indeed, once the rotor reaches the point where the hall sensor is, the position between the output of the harmonic drive and the input (i.e., the rotor) is measured. Because of the position of the magnetic rotary encoder on the output, in each revolution, it rotates by 2.25^{o} degrees (360^{o}/160). This motion generates unique values for each position which are used for the absolute position calculation.

In the start of operation, the rotor moves one turn that equals 2.25^{o} degrees movement in the gear output (joint). During this rotation the hall sensor perceives the magnet attached to the rotor which the controller uses to calculate the absolute position of the joint. Figure 5-16 shows the construction parts of the joint.

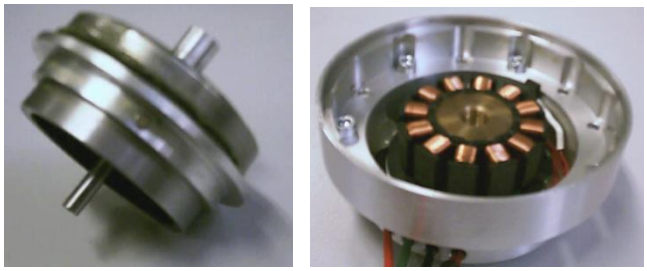


Figure 5-16: Attached magnet of the rotor and hall sensor attached of the frame

5.4.1. Data communication bus

To provide control data for synchronizing the joints, a data communication bus (network) has been used. This network provides a robust data communication with a proper speed and reliability (e.g., SPI[1], CAN[2], LIN[3] or LAN[4]).

An appropriate physical layer for reliable data communication is also necessary. From experience, current loop based physical layers like EIA-422 and EIA-485 are a great choice in this type of applications.

5.5. Processing improvements

The center of mass should be calculated in every control cycle in the robot. This task is time consuming and imposes a heavy load on the central processor. As a result, the central processor cannot be used for other tasks simultaneously. Increasing the speed of the central processor or using a second processor to assist the main processor, are the possible solutions

[1] - Acronym of Serial Peripheral Interface

[2] - Acronym of Control Area Network

[3] - Acronym of Local Interconnect Network

[4] - Acronym of Local Area Network

which cause more energy consumption. Using FPGA for an Application-Specific Integrated Circuit (ASIC) is one of the best solutions for this kind of tasks. In this thesis, the calculation of the center of mass is partially implemented on the FPGA. The main processor puts the data received from the DAU in the specific memory addresses and triggers a signal. After some cycles the ASIC will finish the CM calculation.

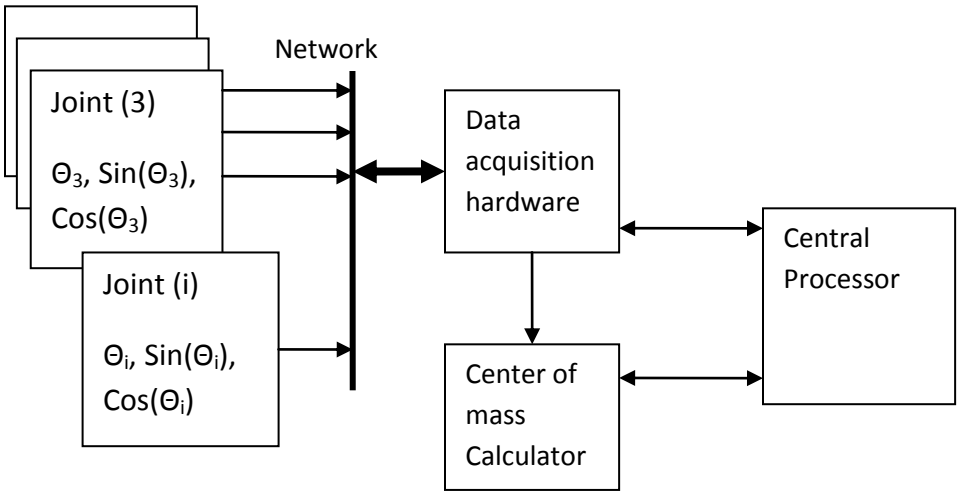


Figure 5-17 : Data flow in optimized control process

Calculating the trigonometric parameters (i.e., Sin (θ) and Cos (θ)) decreases the processing load to find the center of mass in the central controller. Using the parallel processing architecture, individual controllers process the trigonometric parameters in parallel and send it into the central controller. However, in the traditional method, the central controller has to find the trigonometric parameters, for all the joints (i.e., the calculation is not done in parallel). Moreover, using the following method reduces the complexity of the algorithm and makes it possible to be implemented in hardware using Very high integrated Hardware Description Language (VHDL).

5.6. Communication bus interface

Data Acquisition unit regularly exchanges data with the nodes (i.e., joint controllers) in the robot. This communication is broadcasted. In other words, the master (Data Acquisition Unit) writes a message on the bus, and all the nodes receive this message. The nodes analyze the

received data and check the unique ID. If it matches with their ID, they will write the data on the bus (Master Input Slave Output (MISO) line) on the next time frame[5] assigned for data flow from client to the master (Data Acquisition Unit). During this communication only one node writes the data on the bus and the other nodes are in tri-state (otherwise data collision could be happen). Figure 5-18 shows a block diagram of the communication bus based on SPI used in the robot.

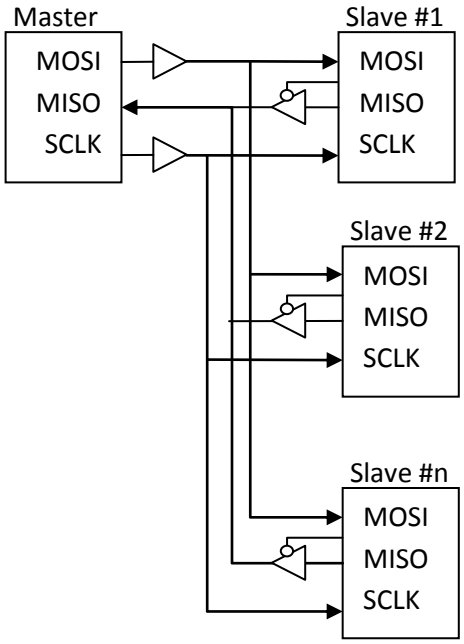


Figure 5-18: Distributed SPI bus used to control the motors

5.6.1. Physical layer

In Archie, like any other tall humanoid robot, the distance between the clients and the Spinal Board (Data Acquisition Unit) is fairly long. Hence, the data can be affected by the noise coming from the environment or the motors of the robot. The high speed communication can also cause some influences in the system and data buses. Thus, a protected communication physical layer is necessary. For this reason, an EIA-422 physical layer is used as the physical layer. The EIA-422 is based on current loop. Figure 5-19 shows a detailed schematic of the EIA-422 bus.

[5] - The time frame consists form 16 clock

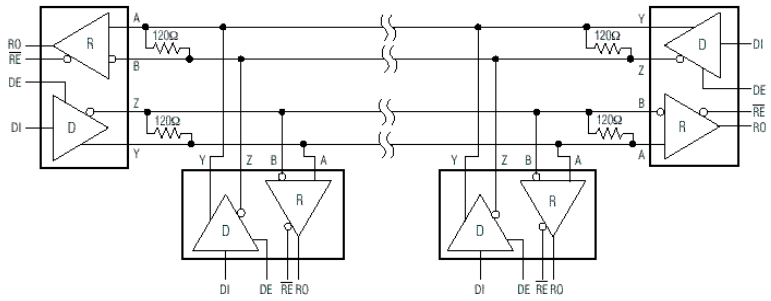


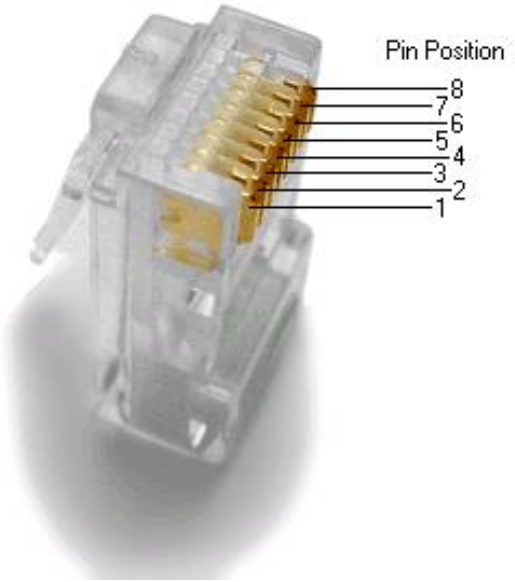
Figure 5-19: Schematic of EIA-422 (from datasheet of SN75ALS180D)

The physical layer should provide the following properties:

- High fan-out: Because of the distributed structure (at least for 16 clients).
- Tri-state mode: Because of the shared bus in receiving data from the clients.
- Robustness, durability and high reliability: Because of the necessity for Real-Time control.
- High speed: To reach a high refresh time (up to 1ms).

The standard category five (Cat-5) cable with twisted pair wires and a shield around the whole cable has been used. Also, the standard Local Area Network (LAN) connectors have been used to connect the cables to the boards (i.e., joint controllers and central controller).

Figure 5-20 illustrates the pin-out for the eight cables that are used in the robot.



Pin description:

1- Master Input slave output (MISO) D+

2- Master Input slave output (MISO) D-

3- Master output slave input (MOSI) D+

4- Master output slave input (MOSI) D-

5- Serial Clock (SCLK) D+

6- Serial Clock (SCLK) D-

7- VCC (9Volt, supply for the controllers)

8- GND

Figure 5-20: The pin-out of the Cat-5 cable used in the robot

The communication bus cables are used in the robot to connect the boards to each other (i.e., daisy chain). Figure 5-21 shows some of the boards that are connected to each other using the daisy chain network.

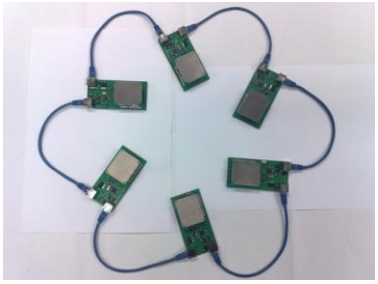


Figure 5-21: Daisy chain network for spreading the communication bus in the robot

5.6.2. Packet structure

A special packet structure has been used to exchange data in the communication bus. The packet structure contains: the unique ID of the node, the type of the data, a command for disabling or enabling the controller and 8-bit data. The packet structure is also able to recognize the noise influence that can change the data bits by using the Cyclic Redundancy Check (CRC). The packet structure uses a 16-bit frame and is depicted in Figure 5-22.

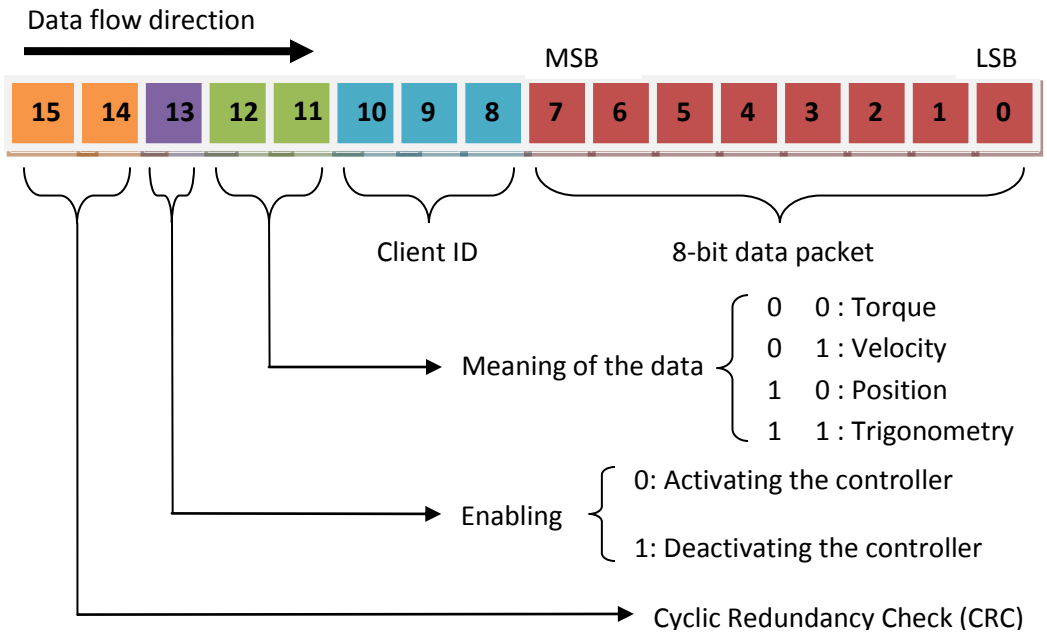


Figure 5-22: SPI communication packet structure

After sending data to a node (e.g., joint controller), the data acquisition unit waits for the same type of data from the specific node in the next time frame. The exchanging operations are implemented in the hardware of the data acquisition unit. It refreshes the entire system in 1ms. In other words, taking the necessary data from the joints, writing on the dual-port RAM, sending the appropriate commands to the joints, is all done in 1ms.

5.6.3. Bit rate calculation

To achieve the 1ms refresh time for the whole robot, the bit rate has to be calculated, in order to select an appropriate physical layer and system design. In general, the bus has the following specifications:

- Total data exchanging time (refresh time): 1ms
- SPI buses: 3
- Maximum clients per bus: 7
- Bits per data packet: 16
- Necessary data packets for data exchanging with one client: 1 send + 1 receive

Therefore: $2 \times 7 \times 16 \times (\text{bit rate}) = 1\ \text{ms} \rightarrow \text{bit rate} = 4.46\ \mu\text{s}$ Equation 5-2

The bit rate is set as the frequency of the serial clock generator (SCLK) for the SPI bus. In addition, this bit rate should be considered to select the physical layer and the clock skew calculation.

5.7. Central controller or spinal board

5.7.1. Control system architecture

The spinal board works as the cerebellum for the humanoid robot and has the following tasks:

- Acquisition of data from the individual joint controllers and sending the calculated information in a constant time.
- Acquisition of data from the inertial measurement unit in a constant time.
- Balancing the whole robot by using the data received from the joints and the IMU.
- Monitoring and handling the failures and errors of the system.
- Sending and receiving data and commands to the higher lever computation system using LAN and wireless communication.
- Supplying power and controlling the energy of the robot.
- Providing Rapid reactions in the robot (like unconscious reactions in Human).

To realize the mentioned tasks for the spinal board, an FPGA with a hardcore implemented PowerPC 405 is used (i.e., Xilinx Virtex 4 FX, XC4VFX12). The chip is implemented with a 16Mbyte SDRAM, USB 2.0 interface, an IrDA[6] interface, and an SD card reader. It also has some other hardware components and ports, all on a small PCB[7] as a daughter board (shown in figure 5-23). It is connected to the Spinal board by a standard DIP40 footprint.

Figure 5-23: Virtex 4 daughter board

[6] - Acronym of Infrared Data Association

[7] - Acronym of Printed Circuit Board

The block diagram of the daughter board is shown in figure 5-24.

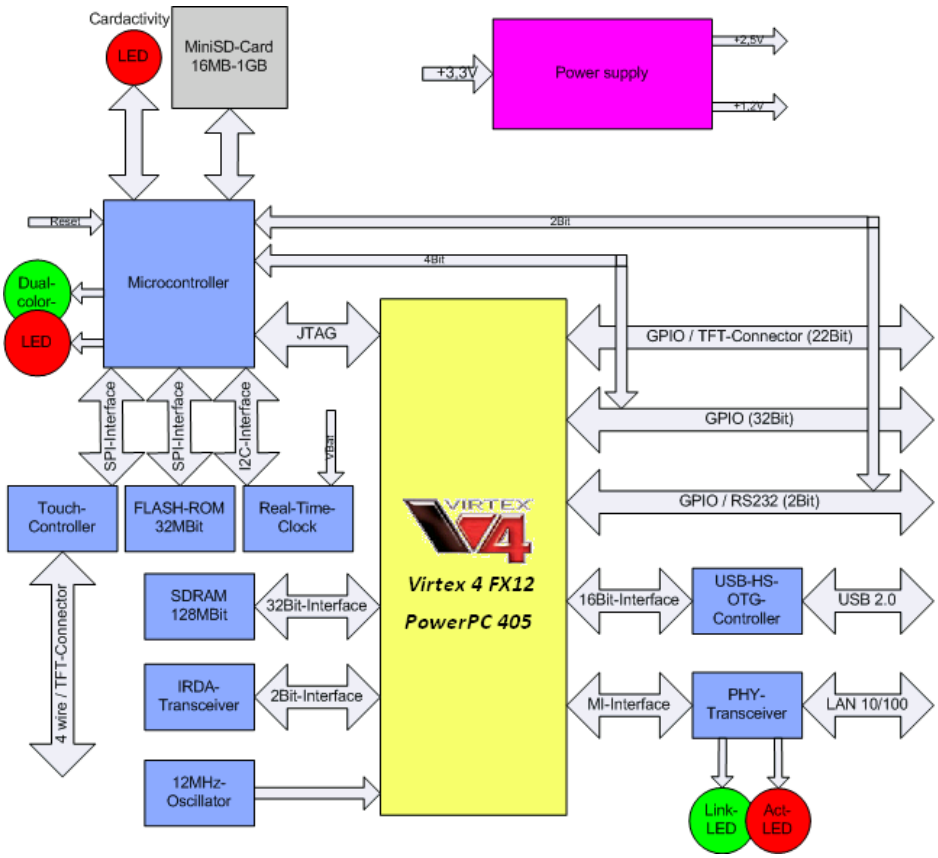


Figure 5-24: Block diagram of the Virtex 4 daughter board

Figure 5-26 shows the daughter board that is mounted on the spinal.

Figure 5-25: Spinal board

Virtex 4 contains the PowerPC 405 and some peripherals, shown in figure 5-26 and 5-27.

Name	Bus I	IP Type	IP Version	IP Classification
ppc405_0		ppc405_virtex4	2.01.b	Processor
plb		plb_v46	1.04.a	PLBV46 Bus
ppc405_0_dplb1		plb_v46	1.04.a	PLBV46 Bus
ppc405_0_iplb1		plb_v46	1.04.a	PLBV46 Bus
SDR_SDRAM_CUSTOM		mpmc	4.03.a	Memory Controller
xps_bram_if_cntlr_1		xps_bram_if_cntlr	1.00.b	Memory Controller
plb_bram_if_cntlr_1_bram		bram_block	1.00.a	Memory
jtagppc_0		jtagppc_cntlr	2.01.c	Peripheral
proc_sys_reset_0		proc_sys_reset	2.00.a	Peripheral
Generic_Ethernet_10_100		xps_ethernetlite	2.01.a	Peripheral
LEDS		xps_gpio	1.00.a	Peripheral
MII_MDC_MDIO_GPIO		xps_gpio	1.00.a	Peripheral
Push_Buttons		xps_gpio	1.00.a	Peripheral
SD		xps_gpio	1.00.a	Peripheral
Generic_IIC_Bus		xps_iic	2.01.a	Peripheral
xps_intc_0		xps_intc	1.00.a	Interrupt Controller
Generic_SPI		xps_spi	2.01.a	Peripheral
SPI_1		xps_spi	2.01.a	Peripheral
SPI_2		xps_spi	2.01.a	Peripheral
SPI_3		xps_spi	2.01.a	Peripheral
xps_timer_1		xps_timer	1.01.a	Peripheral
IMU_unit		xps_uart16550	2.01.a	Peripheral
RS232		xps_uart16550	2.01.a	Peripheral
uOLED		xps_uart16550	2.01.a	Peripheral
clock_72Mhz		dcm_module	1.00.d	IP
clock_72Mhz_90ph		dcm_module	1.00.d	IP
not_gate		util_vector_logic	1.00.a	IP

Figure 5-26: Table of peripherals embedded in the central controller's main processor

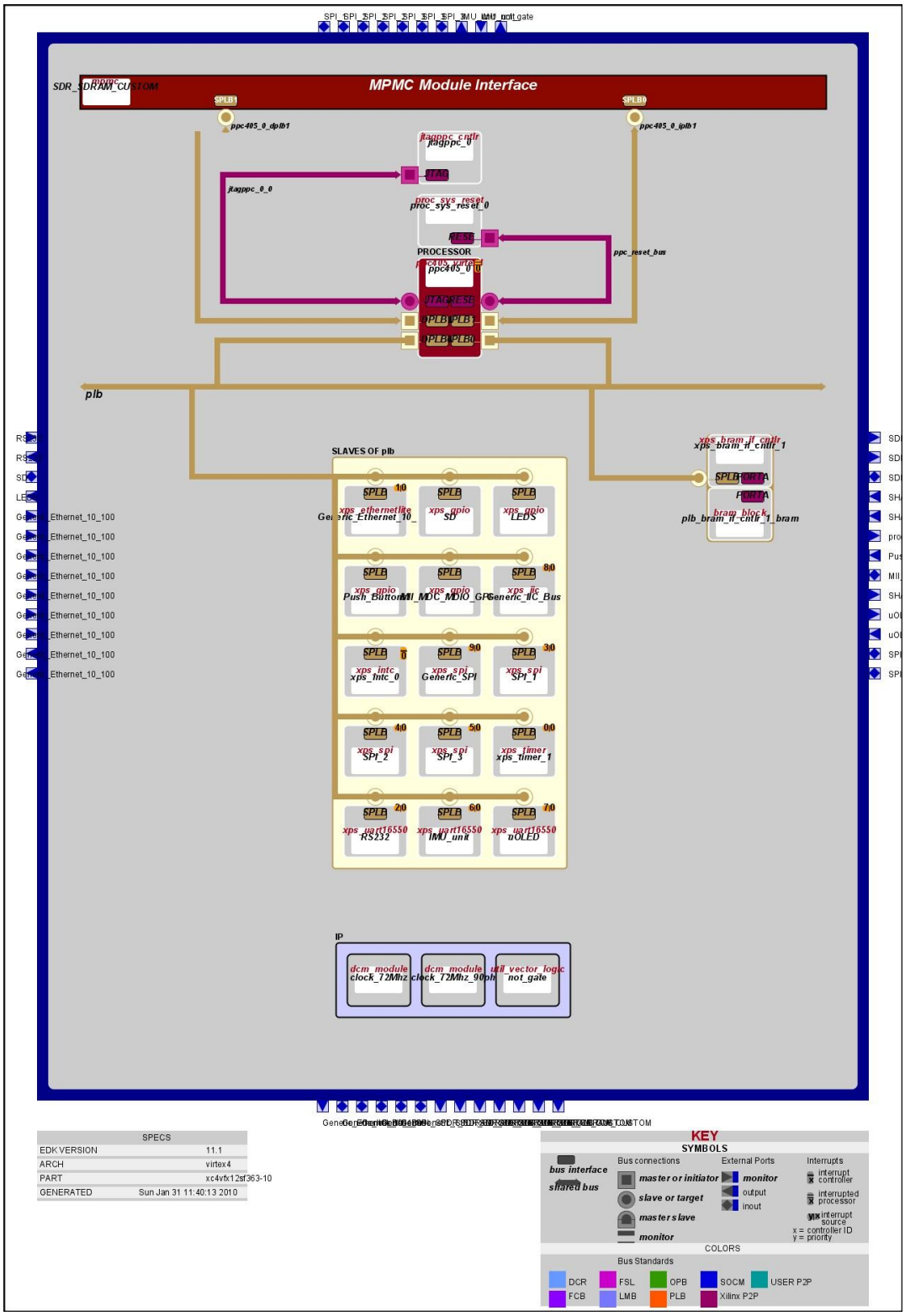


Figure 5-27: The hardware design of the Virtex 4 FPGA used in the central controller (in EDK [8] 10.1)

[8] - Acronym of Embedded Development Kit (Software from Xilinx)

5.7.2. Data acquisition unit

In distributed processing systems, communication between the nodes and the central controller is very important. The communication includes sending and receiving data to and from the nodes. The management of the communication (reading and writing data) imposes a high load on the main processor. As a result, the main processor cannot run the other tasks simultaneously.

Nonetheless, reading and writing data is a repetitively simple task that should be done quickly. This task is similar to the graphic card's RAMDAC[9] which sends the data stored on the graphical RAM to the VGA's analog output sequentially (introduced for the first time by IBM, 1987).

Using software methods to manage the communication will slow down the main processor and may result in losing the constant time to run the communication process. In this thesis, a hardware method has been used to address this problem. The repetitive reading and writing tasks for the communication with the clients is implemented on a piece of hardware. The implementation consists of two parts; the dual ports memory and the stream sequencer. The hardware called Data Acquisition Unit (DAU) is implemented in VHDL[10] language on a part of the FPGA containing the PowerPC 405 hardcore.

The dual-port memory is connected to the main processor via PLB[11] bus. In the other side of the dual-port memory, the stream sequencer is placed. The data stream sequencer is based on the communication packet structure (protocol) and generates an appropriate bit stream which contains all the necessary data. The generated bit stream is transmitted along the communication bus through the physical layer. The data is received by all the nodes and checked by their unique ID.

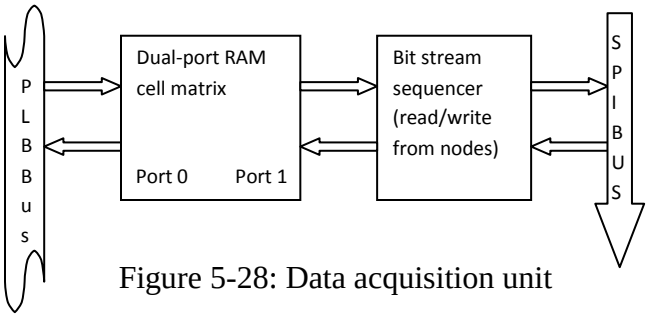


Figure 5-28: Data acquisition unit

[9] - Acronym of Random Access Memory Digital-to-Analog Converter

[10] - Acronym of Very high integrated Hardware Description Language

[11] - Acronym of Processor Local Bus

5.7.3. Operation system

Archie's central controller uses the Linux operating system (OS) based on the 2.6 kernel[12] version. This OS provides all the time management for the tasks and balancing computations. The Linux operating system used in Archie (called Archie Linux) is a standard 2.6 kernel ported on the PowerPC 405 processor (Embedded Processor inside the FPGA).

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus	IP Type	IP Version
ppc405_0's Address Map								
SDR_SDRAM_CUSTOM	C_MPMC_BASEA...	0x00000000	0x00FFFFFF	16M	SPLB0:SPLB1		mpmc	4.03.a
ppc405_0	C_IDCR_BASEAD...	0B0100000000	0B0111111111	256	Not Connected		ppc405_virt...	2.01.b
Generic_Ethernet_10_100	C_BASEADDR	0x81000000	0x8100FFFF	64K	SPLB	plb	xps_etherne...	2.01.a
SD	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	plb	xps_gpio	1.00.a
Push_Buttons	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	plb	xps_gpio	1.00.a
MII_MDC_MDIO_GPIO	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	plb	xps_gpio	1.00.a
LEDS	C_BASEADDR	0x81460000	0x8146FFFF	64K	SPLB	plb	xps_gpio	1.00.a
Generic_IIC_Bus	C_BASEADDR	0x81600000	0x8160FFFF	64K	SPLB	plb	xps_iic	2.01.a
xps_intc_0	C_BASEADDR	0x81800000	0x8180FFFF	64K	SPLB	plb	xps_intc	1.00.a
xps_timer_1	C_BASEADDR	0x83C00000	0x83C0FFFF	64K	SPLB	plb	xps_timer	1.01.a
Generic_SPI	C_BASEADDR	0x83C12000	0x83C1207F	128	SPLB	plb	xps_spi	2.01.a
SPI_2	C_BASEADDR	0x83C14000	0x83C1407F	128	SPLB	plb	xps_spi	2.01.a
SPI_3	C_BASEADDR	0x83C18000	0x83C1807F	128	SPLB	plb	xps_spi	2.01.a
SPI_1	C_BASEADDR	0x83C1C080	0x83C1C0FF	128	SPLB	plb	xps_spi	2.01.a
uOLED	C_BASEADDR	0x83E00000	0x83E0FFFF	64K	SPLB	plb	xps_uart165...	2.01.a
RS232	C_BASEADDR	0x83E20000	0x83E2FFFF	64K	SPLB	plb	xps_uart165...	2.01.a
IMU_unit	C_BASEADDR	0x83E40000	0x83E4FFFF	64K	SPLB	plb	xps_uart165...	2.01.a
xps_bram_if_cntlr_1	C_BASEADDR	0xFFFF8000	0xFFFFFFFF	32K	SPLB	plb	xps_bram_if...	1.00.b

Figure 5-29: Memory map of the central controller's main processor

Running the Linux operating system requires minimum peripherals (i.e., at least an interrupt controller, system timer and etc.). All the peripherals are placed in a unified memory map that is shown in figure 5-29. Following configurations are part of the Device Tree Blob (DTB). The device tree blob is used to compile the kernel of the operating system (Linux). In the end the firmware (the FPGA hardware configuration file), the kernel image (the image file of the compiled Linux kernel) and the device tree blob are used to execute the operating system.

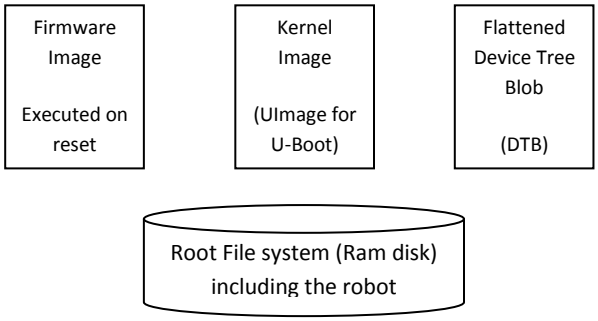


Figure 5-30: Prepared files for running the operationg system

[12] - The latest Linux kernel was released on 23 December 2003

5.7.4. Booting up mechanism of the FPGA and the Linux

The SD memory card inserted in the memory card reader on the Virtex 4 daughter board, includes, two files. One of them is a hardware set bit stream and the other is an image file. The first file contains all the configurations of the FPGA (includes hardware design) and a loader program. This stream is programmed on the FPGA using a microcontroller. The microcontroller is used as the BIOS[13] and starts up on the power up once the power is plugged to the system. The microcontroller starts to read the bit stream from the Secured Digital (SD) memory card and pumps it into the FPGA using the JTAG[14] bus. After finishing programming the bit stream into the FPGA, the created design executes the loader program. The loader program is pre-programmed by the hardware configuration bit stream inside the program memory of the processor. The loader program reads the image file and decompresses it into the SDRAM[15]. During this operation, the display on the spinal shows a progress bar. After decompressing the kernel, the Memory Management Unit (MMU) is turned on. In this moment, the kernel starts to manage the hardware and executes the robot's control software. The flow diagram in figure 5-31 shows the entire process.

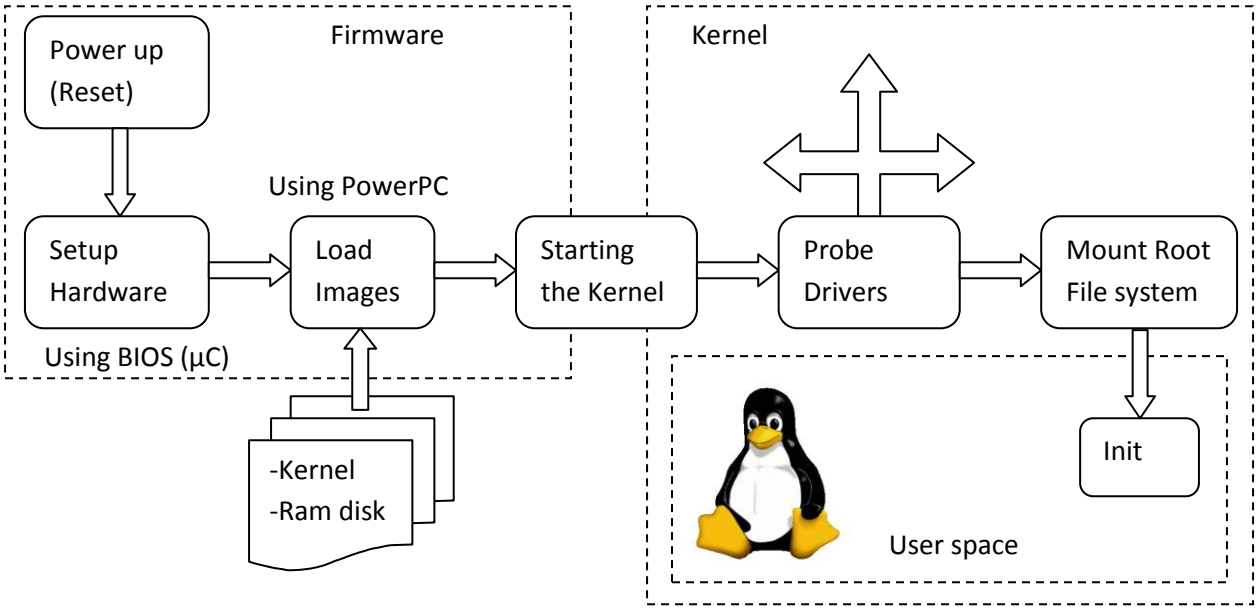


Figure 5-31: Booting the embedded system on chip and Linux flow

[13] -Acronym of Basic Input Output System

[14] -Acronym of Joint Test Action Group which is a type of programming interfaces

[15] -Acronym of Synchronous Dynamic Random Access Memory

5.8. Energy management

One of the most important operational factors in mobile robots is energy. Archie is a mobile robot and it should operate for at least one hour per each full-charged battery.

To get a better result, the efficiency should be well considered. The design should consider minimizing the energy which is lost in the electronic and mechanical components. The power supply plays a big role in this criterion. In addition, the battery charger circuit should be designed for the minimum necessary time to fully charge the battery. Table 5-3 shows all the components in the robot, the required voltage and the maximum wattage of them.

Component	**Voltage**	**Maximum wattage**
Brushless motors	28 volt	50 watt x (4+4+1) = 450 watt
DC motors	24 ~ 28 volt	19 watt x (2+2+3) = 133 watt
Brushless based joint Controller	7 ~ 12 volt	1~2 watt x (4+4+1) = 9 ~ 18 watt
DC based joint Controller	7 ~ 12 volt	1 watt (2+2+3) = 7 watt
Spinal Board + FPGA	9 volt	2 watt + 3 watt = 5 watt
Servo Motors (RX 64)	18 volt	21.6 watt x 13 = 280 wall
Total wattage:		450 + 133 + 18 + 7 + 5 + 280 = 893 watt

Table 5-3: Components in the robot and the required voltage and maximum wattage

What is described in table 5-3 is for the worst case situation which hardly happens.

5.8.1. Batteries

There are two Battery packs in the robot as the energy sources. The battery pack has the following specifications (also shown in figure 5-32):

Chemical type: Lithium Ion

Voltage: (4.2 v ~ 3.6 v) x 7 cell = 29.4 v ~ 25.2 v Equation 5-3

Amperage per hour: 6.1 A/hour

Figure 5-32: Battery pack used in Archie

The stored energy in the power source is:

2 x 4 x 7 x 6.1 = 341.6 watt/hour Equation 5-4

A humanoid robot rarely uses all the motors with the maximum power. There are some experimental founded relations that say that only a quarter of the motors are usually under full power work. By using this assumption, the quarter of the maximum power used by the motors plus the energy of the other control equipments in the robot is:

(450 + 133 + 280) / 4 + (18 + 7 + 5) = 215.5 + 30 = 245.5 watt Equation 5-5

Therefore, the expected operation time for the robot with a full-charged battery is:

341.6 / 245.5 = 1.4 hour = 1:24 that means 1 hour and 24 minutes Equation 5-6

5.8.2. Power supply

The Power supply is an electronic unit that provides different voltages for the components of the system. The batteries that are used in the robot are designed to supply 29 to 25 volts (29.4 in full charge and it decreases to 25 during using the battery). The motors used in the robot (expect the servo motors) work on the same voltage range; Hence, there is no need for additional voltage regulating circuit.

The servo motor (RX 64) uses switching power supply with 300 watts maximum output power this converts a voltage range of 20 to 30 to a regulated 18 volt output.

For the control equipments, there is a second switching power supply with 50 watt maximum output power that converts a voltage range of 20 to 30 to a regulated 8 volt output. This power rail is used for the whole control equipments in the robot.

The Spinal board uses a linear voltage regulator to convert the 8 volt input to the 5 volt output. The 5 volt rail is used to supply other peripherals on the spinal board. The FPGA uses a miniature switching power supply on the spinal board that converts the 5 volt input to the 3.3 volt output which is also used in the wireless communication module and the OLED[16] display (µOLED-160-G1, 4D systems 2008) on the spinal board.

The 8 volt rail goes directly in the communication bus cables that are based on standard Cat-5 cables and are used on the joint controllers. The joint controllers have a linear regulator which converts the 8 volt to the regulated 5 volt. The 5 volt is used for the joint controller processors and the encoders.

[16] - Acronym of Organic Light Emitted Diode

5.9. Normal operation flowchart

Figure 5-33 illustrates the operation flowchart for a normal operation of the robot.

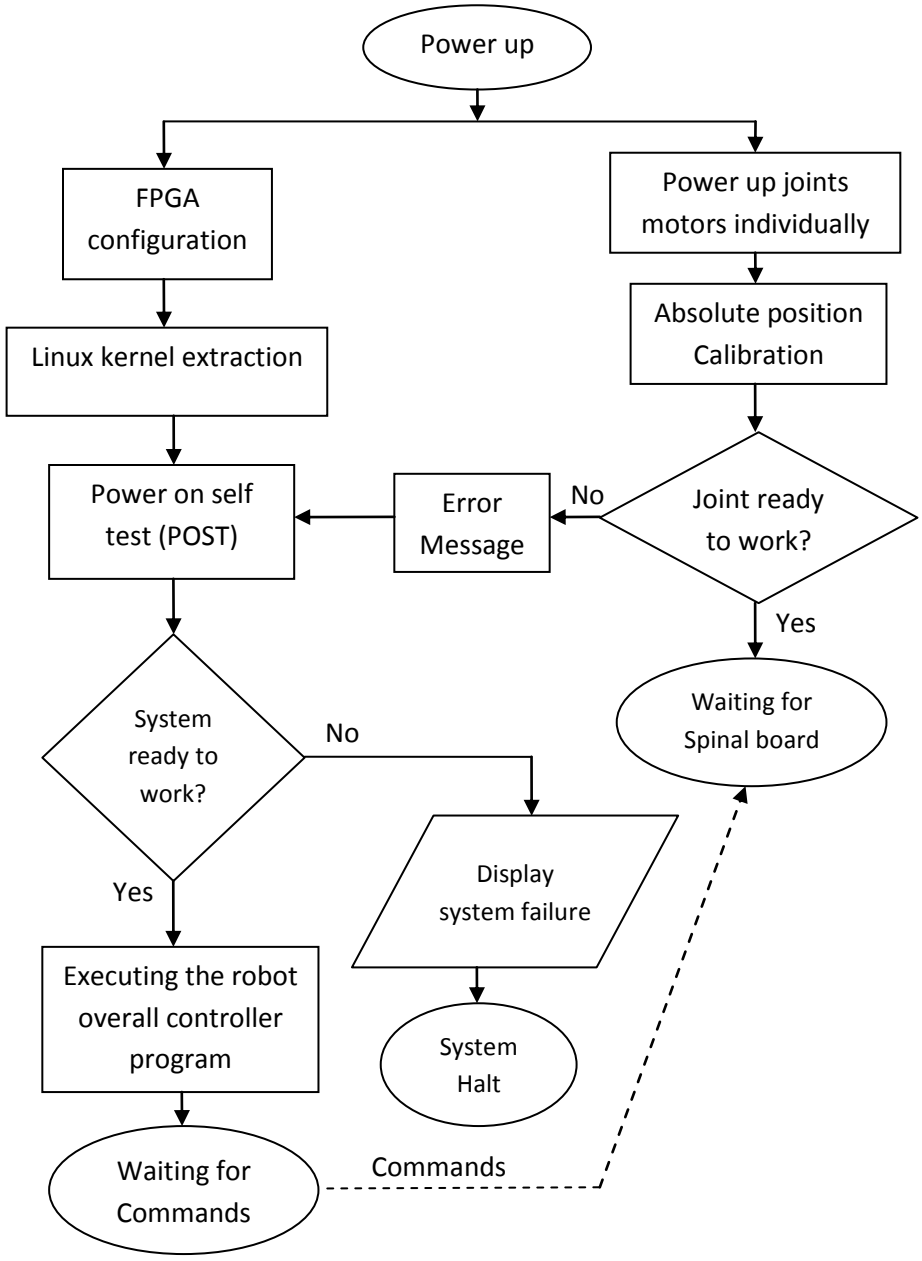


Figure 5-33: Flow chart for notmal operation

5.10. Applications on the robot

5.10.1. Motion planner

After running the balancing operation in the spinal board, the robot is ready to execute higher level commands. The higher level commands can be issued by different motion planners.

For example, a motion planner can be used for playing soccer or other purposes.

The Motion Development Planner (MDP) is a type of motion planner that is used for motion development and finding motions in the robot.

Archie uses a motion development planner written with C++ in Linux. Figure 5-34 shows a screen shot of this motion planner.

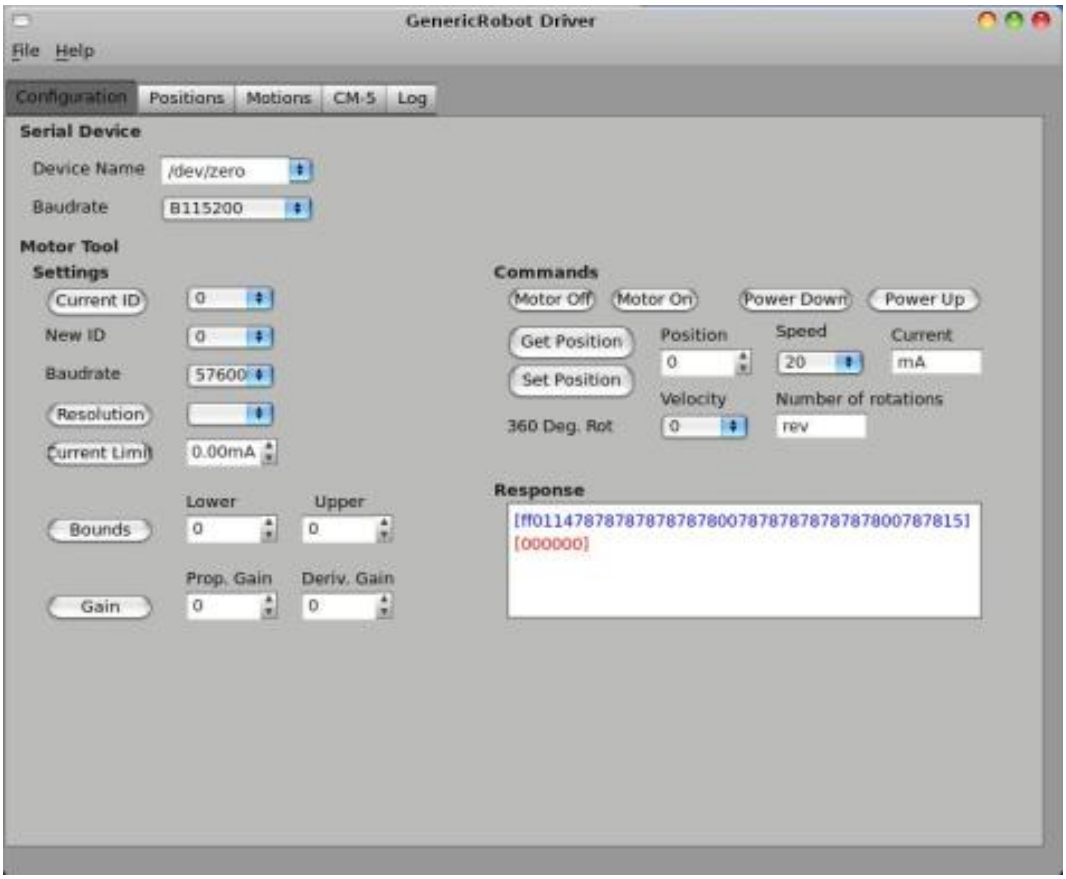


Figure 5-34: Screen shot of the motion planner configuration tab

In the configuration tab of the motion planner, there are some parameters that can be set (figure 5-34):

- Communication device: It can be either serial port or LAN.
- Baud rate: The desired speed for sending data to the spinal board.
- Initial settings: The initial current limit for the joint controllers.
- Offset setting: Setting the offset for the joints in order to calibrate.
- Testing the joints: Testing the Joints individually and monitoring their data.

The second tab in the motion planner is used to develop the positions. Figure 5-35 shows a screen shot of this tab.

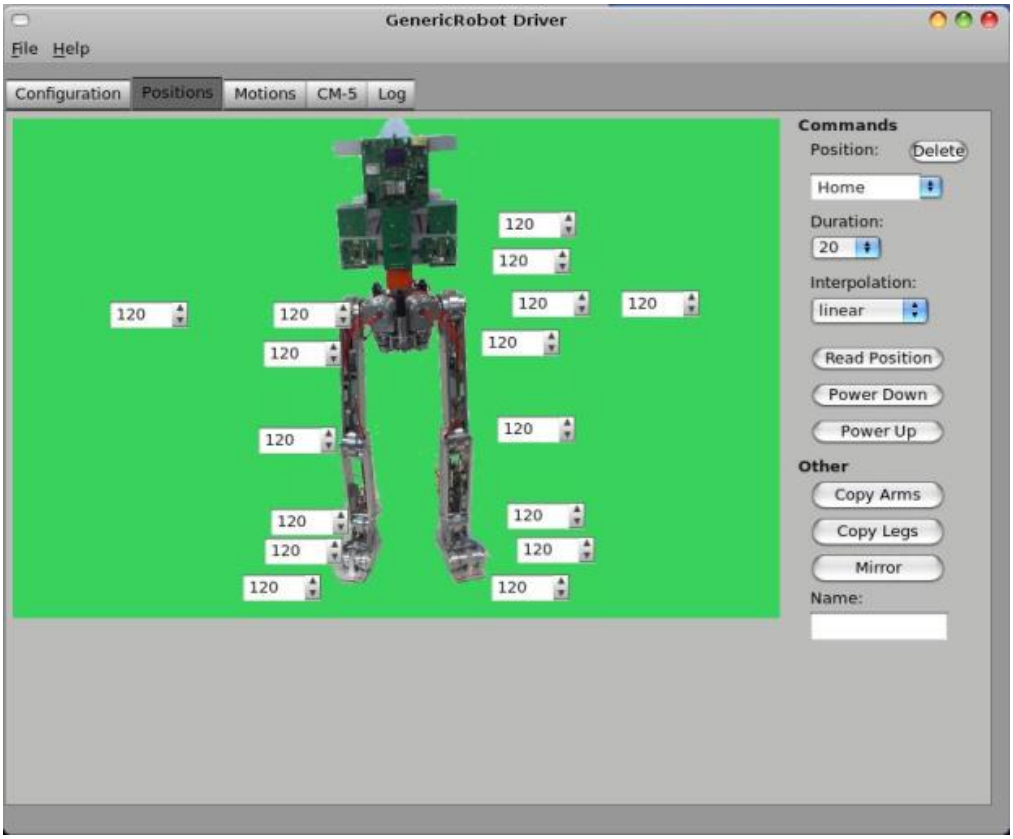


Figure 5-35: Position development tab screen shot

The Position development tab develops motions and stores them in the robot. After storing the positions, the robot is able to go to a specific position by calling that position. When the robot stays in a position, the spinal board tries to hold the balance by processing the data received from the feedback data from the Inertial Measurement Unit (IMU) and the positions of the other joints.

In other words, when a movement is applied to a joint, and the balancing is enabled, the spinal board issues appropriate commands to the other joints to hold the overall balance of the whole robot.

The resulting positions are stored as commands for the robot. By executing a sequence of positions to the robot, motion is performed. Figure 5-36 shows a screen shot of a tab that provides this ability in the robot.

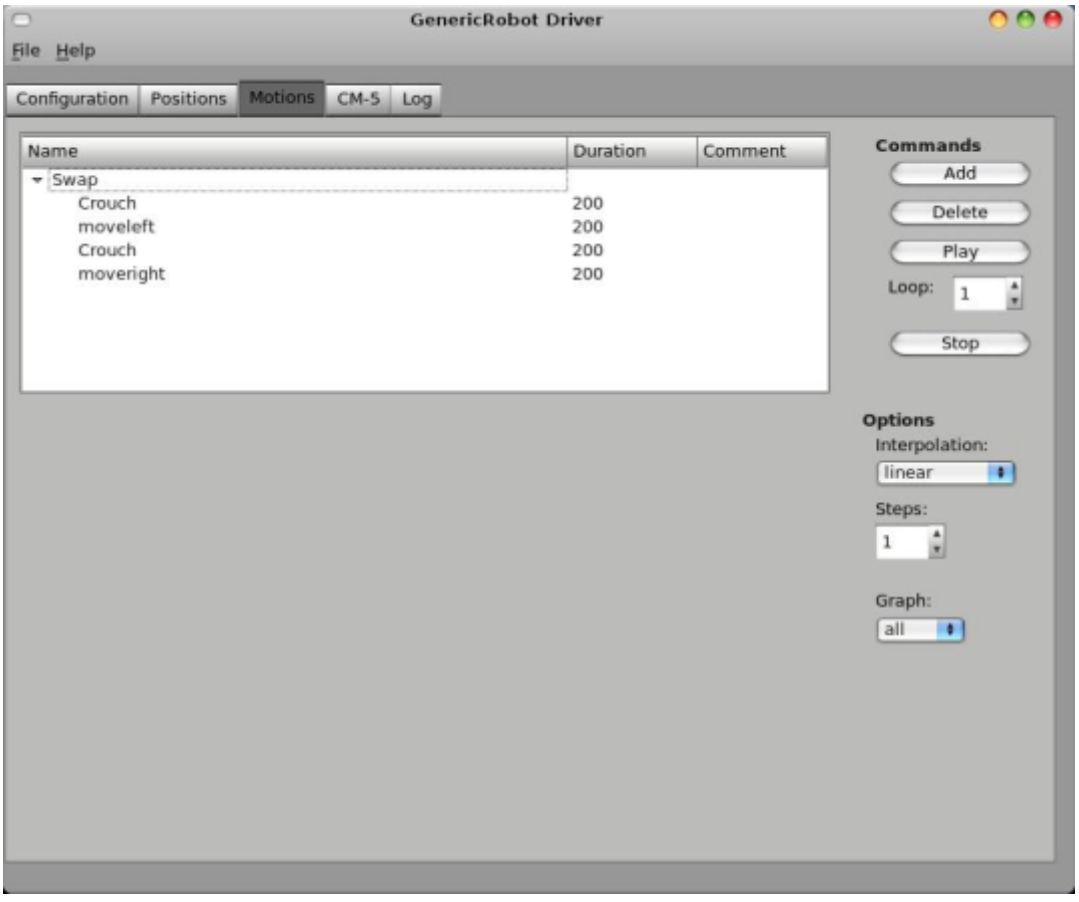


Figure 5-36: Motion development tab screen shot

For example, using three different positions and issuing them sequentially, the robot is able to have a dynamic walk which is considered as a motion of the robot.

The last tab is for logging all the data which is send from the motion development planner to the robot (Figure 5-37).

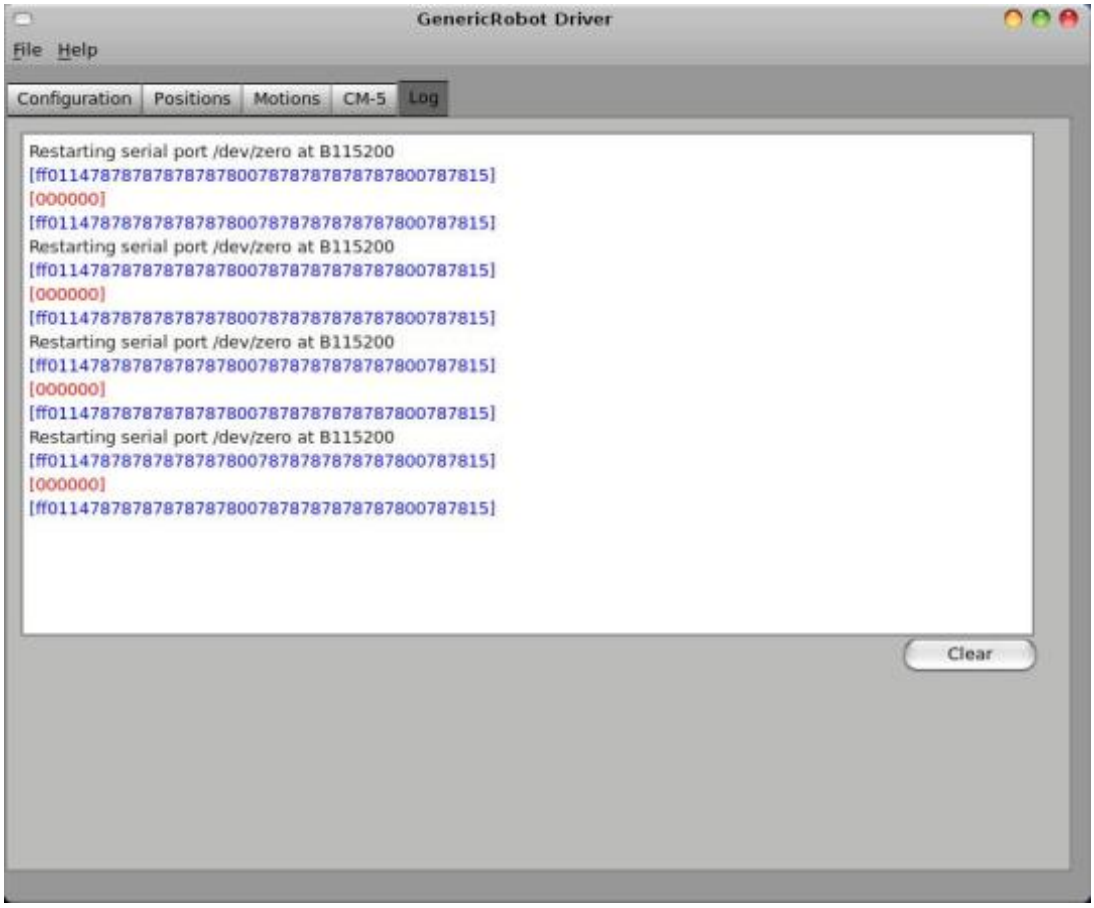


Figure 5-37: Motion development planner logging tab screen shot

5.10.2. Joint offset assignment

Each joint has a certain point as zero. When all the joints are replaced on the zero position, the robot stands in a ‘T’ form (Figure 5-38).

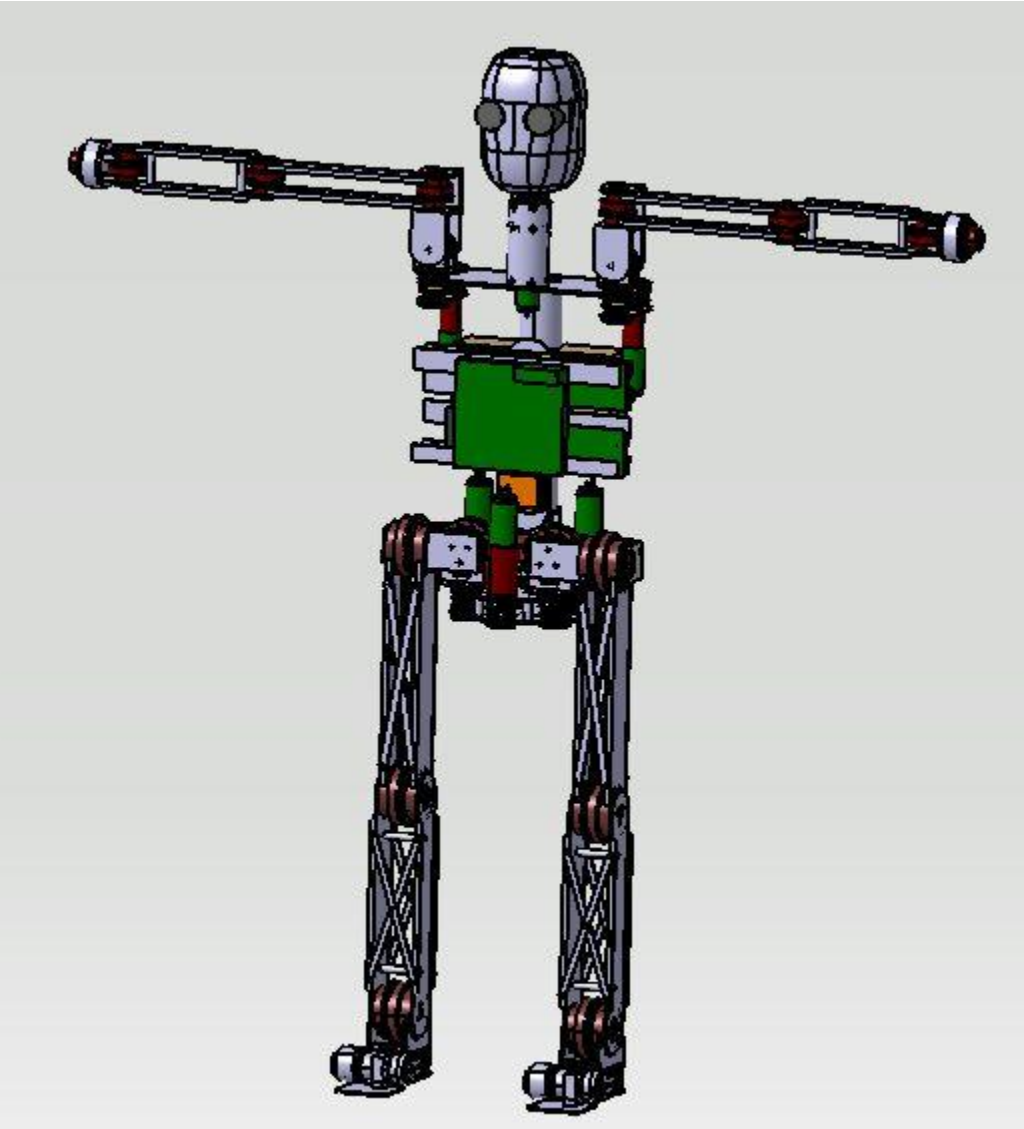


Figure 5-38: Simulation of Archie standing in ‘T’ form

The T form position is used for calibration. To do so, all joints move to the proper position to have the whole robot standing in the ‘T’ form. Then, all settings are stored in the joint controllers.

Calibration is not always necessary and is only required after mechanical reassembling. However, after a long time of operation it is necessary for the robot to be recalibrated again.

5.11.Robot simulator

The simulator used for the robot is developed in the SimMechanics toolbox of Matlab-Simulink. The simulator is prepared in order to develop the motions and predict the real results of the robot before applying them to the real robot.

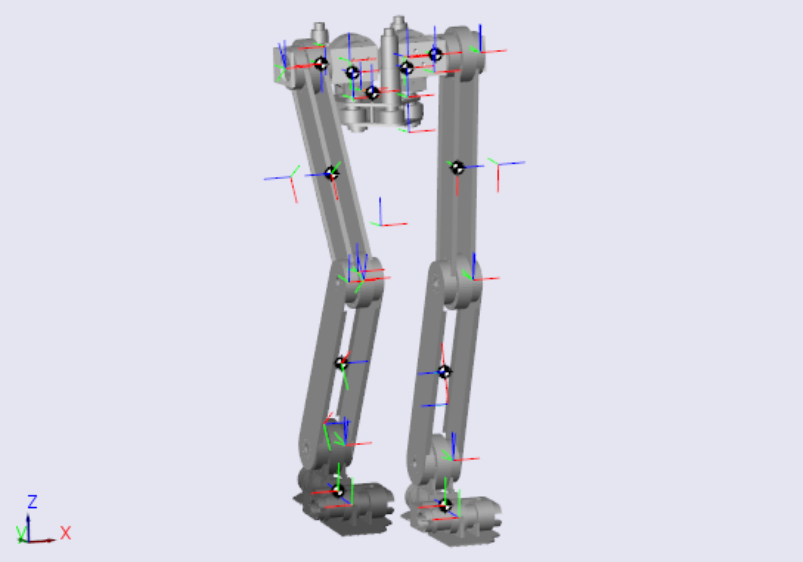


Figure 5-39: Simulation of Archie's lower body

One of the main benefits of using a simulator is the possibility of monitoring the trajectories caused by the movements of the robot. Figure 5-40 shows the block diagram in the Matlab Simulink of the robot simulator.

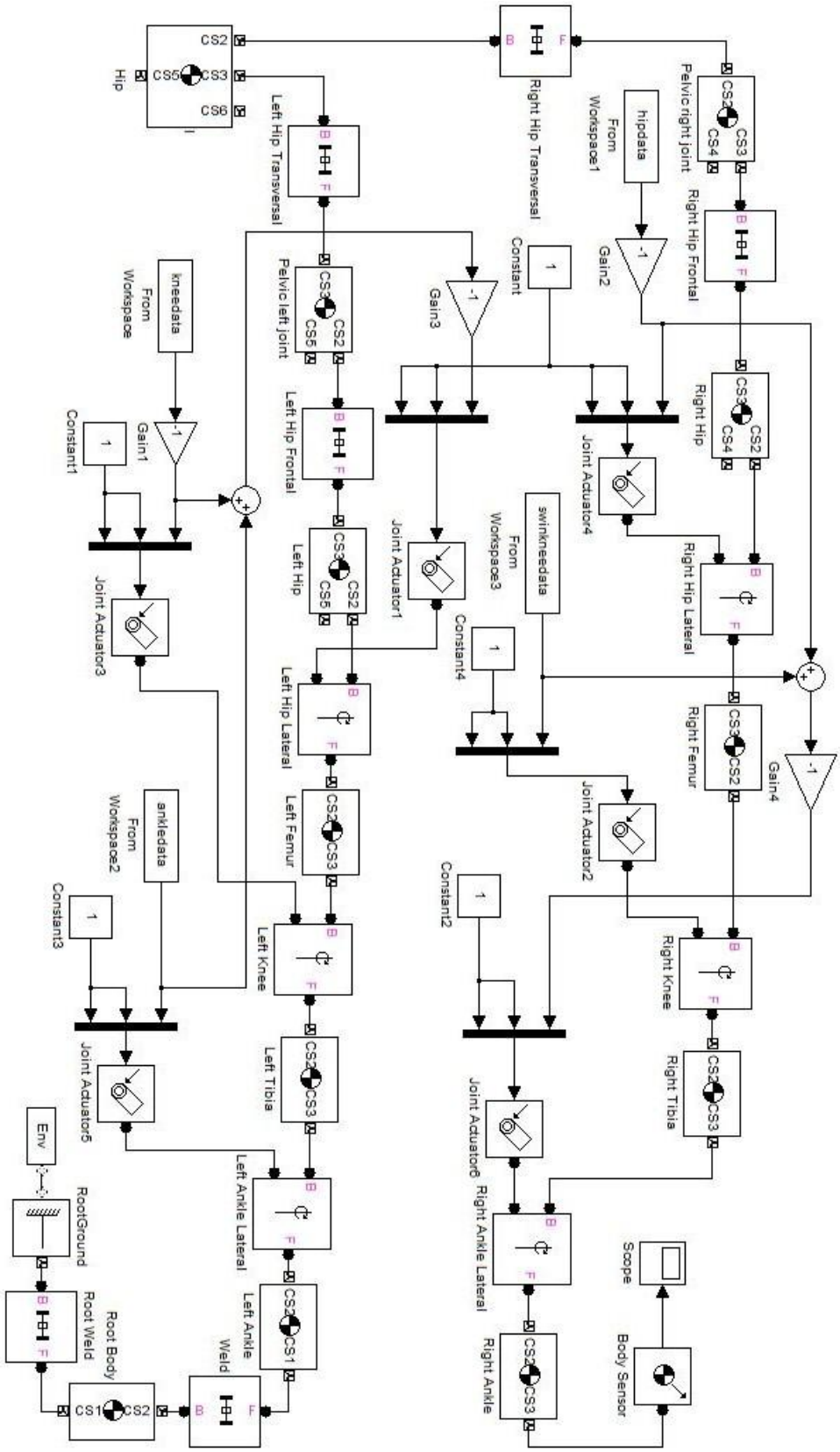


Figure 5-40: Archie's lower body simulation in Simulink

Chapter 6

6. Tests and Results

In this chapter the results of the practical tests on the robot are presented. The tests are performed in three different levels. In the first level a single joint controller is tested. The joint controller test consists of testing the performance of the velocity controller and the position controller (more description in section 6.1). In the second level which is an expansion of the first level, a combination of two joints (knee and lateral hip joint) is used to test the robot's leg. In the Third level the robot's legs (left leg and right leg) are attached to the torso of the robot and the robot is tested for basic walking sequence.

The trajectories obtained from the tests on the robot are compared with the trajectories acquired from the robot's simulator. The comparison is used for testing the prediction capability of the robot's simulator and performance of the robot's control system.

6.1. Joint controller test

In this section, a single joint of the robot is tested separately. For this test, the standard Motion Monitor Toolset (Elmo MC Composer, ver. 2.19 July 2008) is used. The joint controller is tested in a situation that is counted as the worst case of system instability. In this situation the lateral hip joint is used to move the robot's leg from $0°$ to $45°$ (Figure 6-1).

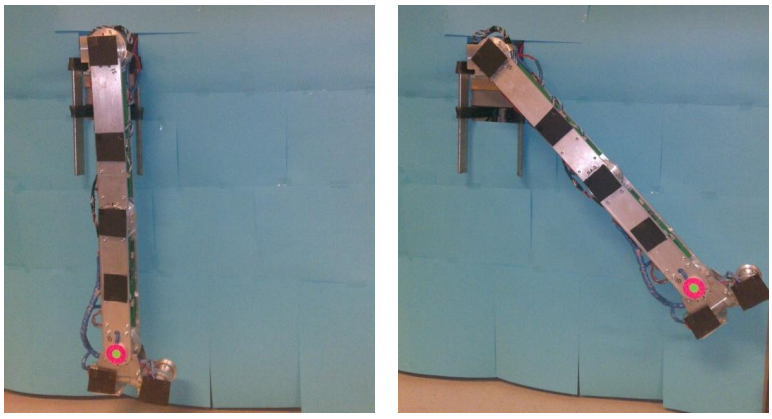


Figure 6-1: Robot's leg used for joint controller test

In the first movement (0° to 45°), the joint controller is affected by the gravitational forces of the leg that increase gradually by the growth of the lateral hip joint's angle ($m_{leg}\, g.Sin\theta_{Lateral\ \ Hip}$).

The test is performed on different velocities from the trajectory planner. Figure 6-2 shows the Simulink block diagram for the single joint test simulation. In this simulation, one leg of the robot is performed in swinging phase which means that the leg is attached from the hip to the simulation environment root. The joint that is on evaluation is the lateral hip joint (Right leg lateral hip joint). Any other joints of the leg in the simulation are supposed to be fixed without movement (welded joints).

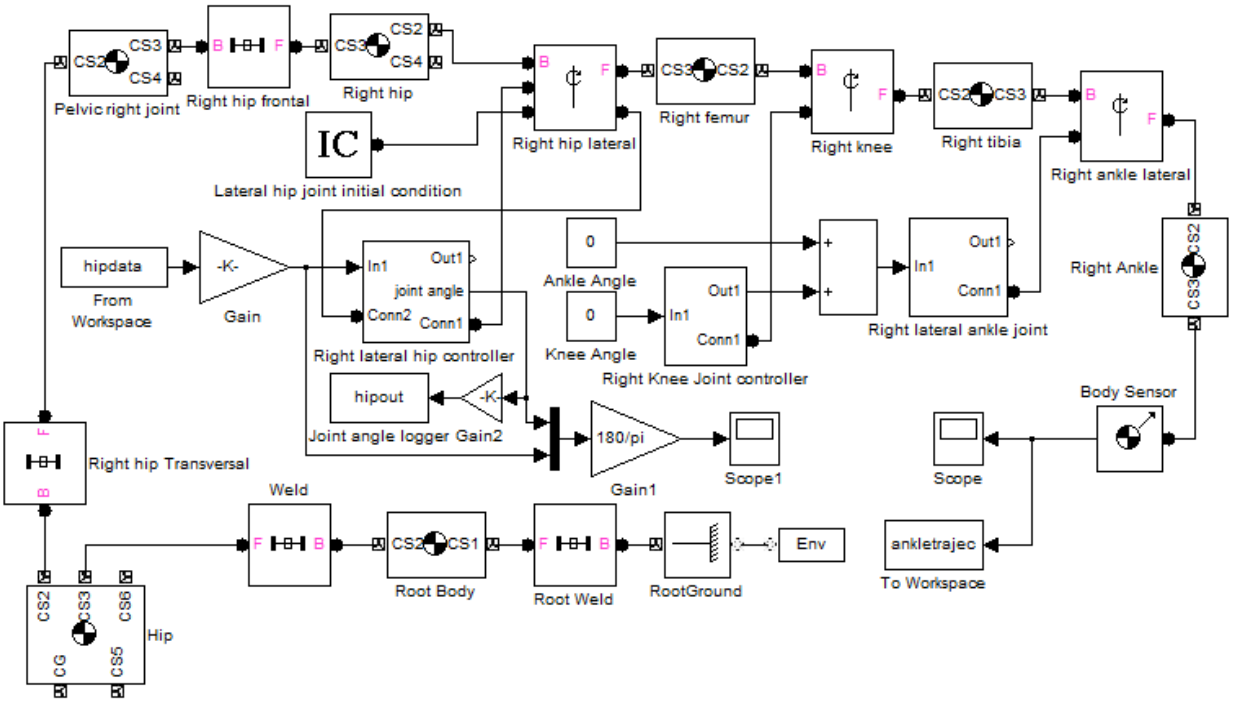


Figure 6-2: Simulink block diagram for single joint test

The simulation block diagram (shown in figure 6-2) contains a joint controller block which is extracted from the right lateral hip controller block and it is shown in figure 6-3. The block diagram of the controller includes the gravity effect that causes non-linearity in the system model. The gravitational effect is applied by adding the vertical component of the leg multiplied by the weight and divided by the joint's gear ratio. The system plant is thought as a continuous-time model. Thus, the input and output of the system plant model is connected by zero-order hold blocks to the other parts of the simulation (the other parts of the simulation are based on the discrete-time model).

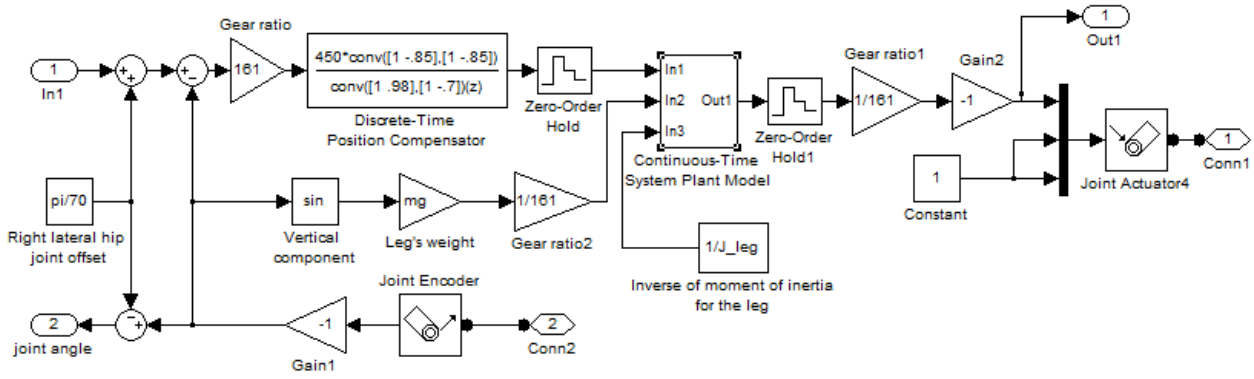


Figure 6-3: Extraction of the joint controller

The extraction of the system plant block is shown in figure 6-4. The model contains all the simulation parameters for the electric motor used in the joint as actuator. The model starts with the rotor voltage as input and converts it into current by the first integrator. The motor is designed for an allowed current range which is applied to the simulator by a limiter block. The result (and limited) rotor current is multiplied by the electromotive force constant (K_t) of the motor and resembles the generated torque of motor. By adding the gravitational force and the damping ratio caused by friction of the mechanical plant, the actuator torque is resembled. By multiplying the total inverse of the moment of inertia to the actuator torque, the result is the angular acceleration of the joint. The angular acceleration is then converted to angular velocity by using the second integrator. The motor's mechanical construction restricts the angular velocity of the motor in a definite range, which is resembled by the second limiter block. Finally, the angular velocity is turned into the motor's angle by the third integrator.

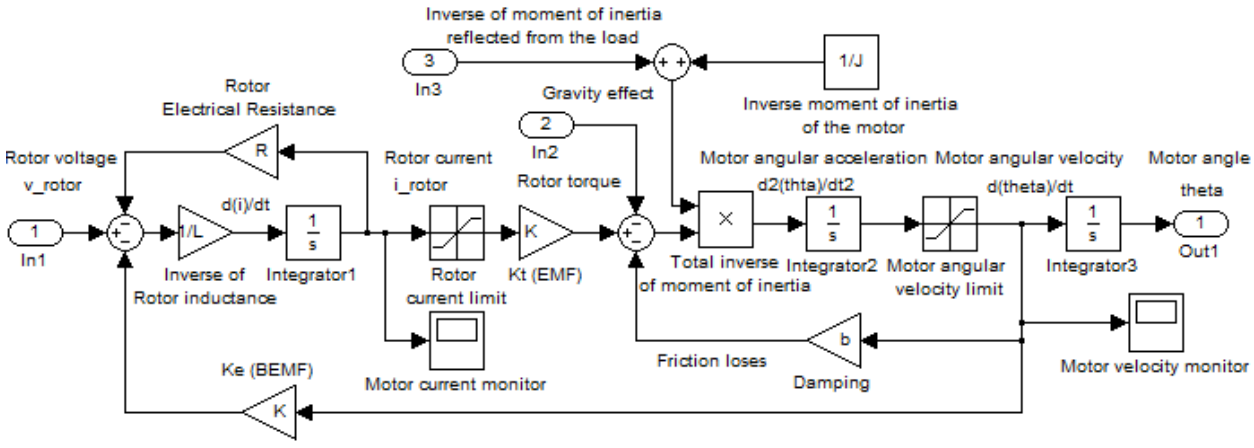


Figure 6-4: Continuous-time system plant model

In figure 6-5 the robot's right leg is illustrated in SimMechanics Simulator output.

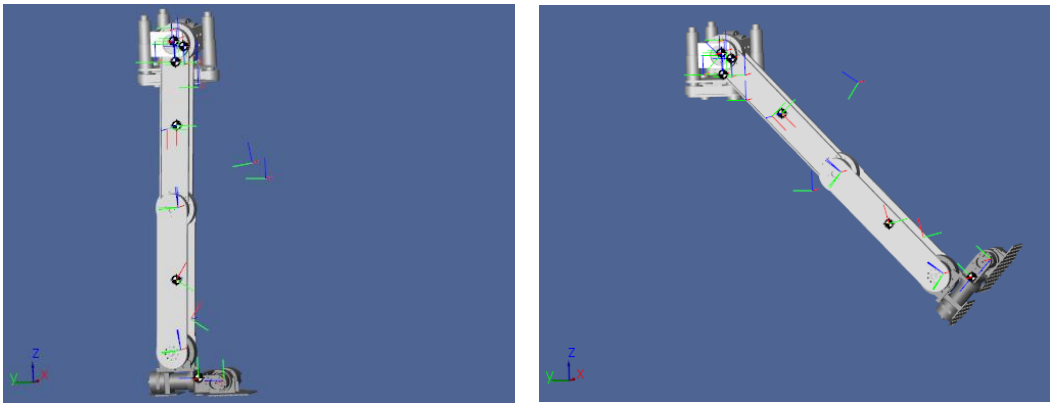


Figure 6-5: Simulation of the robot's leg used for joint controller test

In this test, the swinging robot's leg moves from vertical stationary position to 45° angle position. During the test, the gravitational effect is reflected to the joint's actuator gradually which causes non-linearity of the system plant model. The test is executed on the real robot and the robot's simulator.

The joint controller limits the velocity of the motor on the desired value, in order to control the necessary time used for the movement. Controlling the traversing time gives the possibility to synchronize the joints for combinational movements (e.g., human gait imitation).

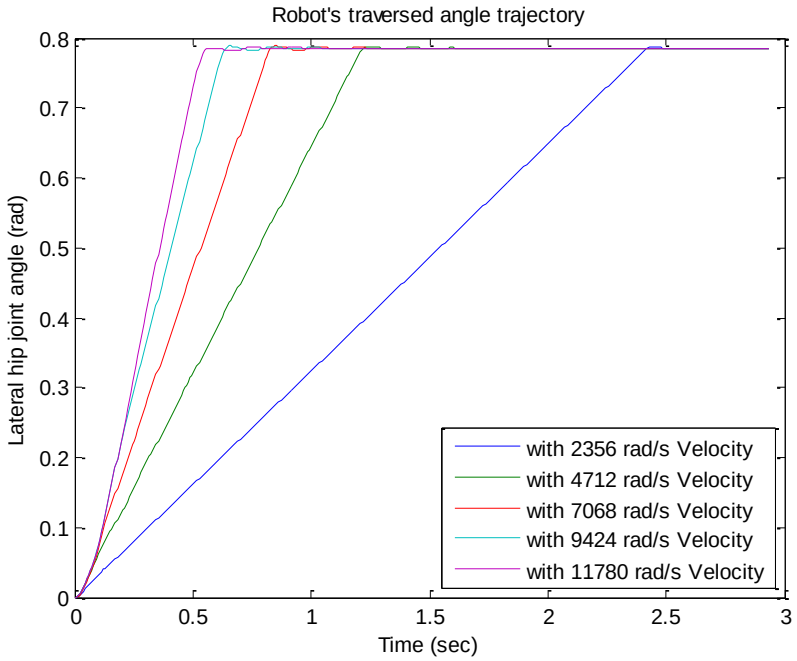


Figure 6-6: Traversed angle trajectory by the real robot with different movement velocities

The angular trajectory error caused by the real robot from the movement of the hip joint with different traversing velocities is depicted in figure 6-7.

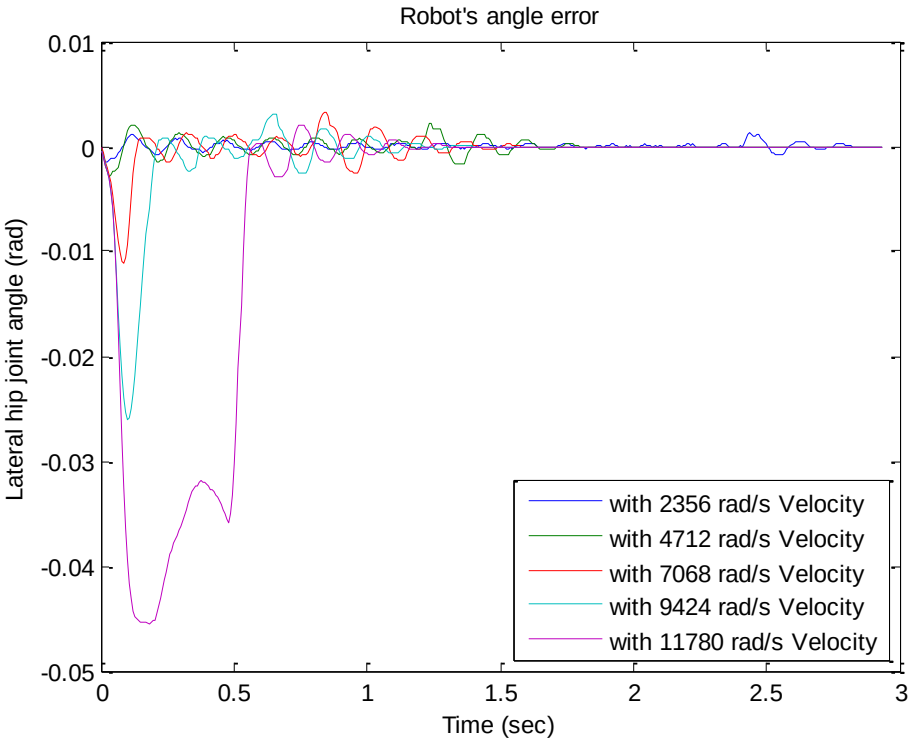


Figure 6-7: Angular trajectory error caused by the real robot tested with different traversing velocities

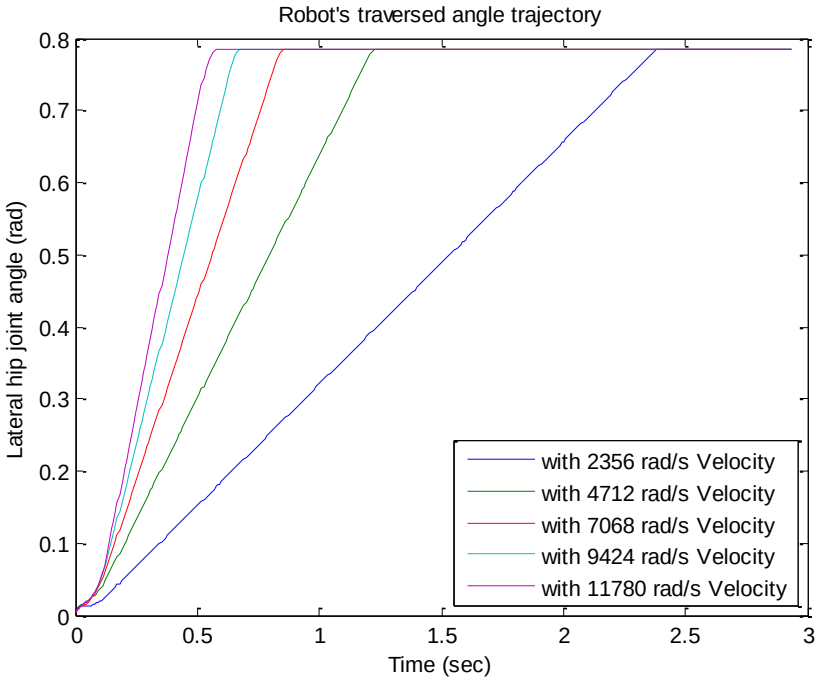


Figure 6-8: Traversed angle trajectory by the robot simulator, with different movement velocities

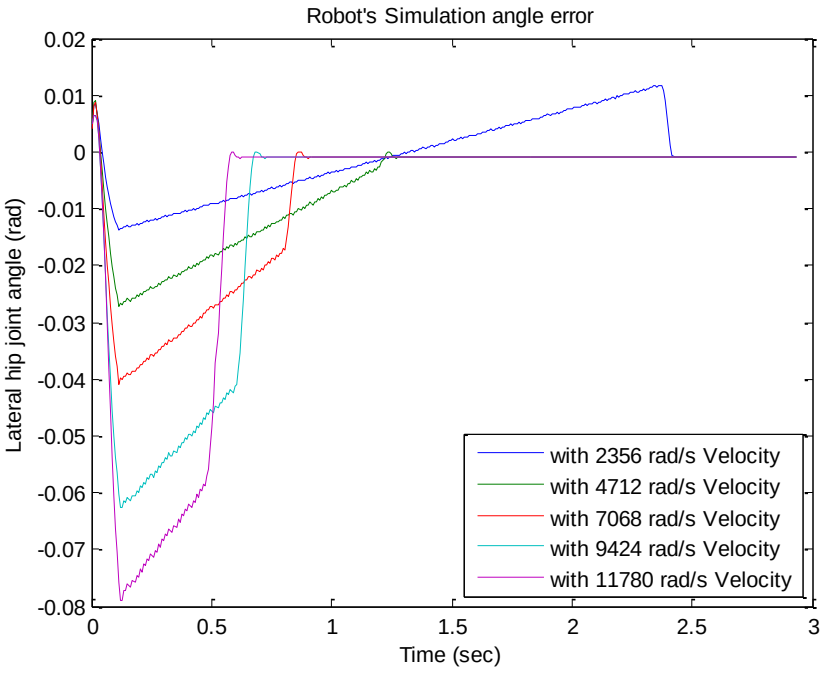


Figure 6-9: Angular trajectory error caused by the robot simulation tested with different traversing velocities

Figure 6-7 illustrates how in the $0°$ to $45°$ test, the error arises in the beginning of the motion and it gradually decreases on time. When increasing the velocity, the error is proportionally increased. This is shown by the purple-color line that has the highest traversing velocity, this trajectory showed the highest error of all. In addition, for this velocity the error stays during the whole period of movement on a fairly constant range. On the other hand, when using smaller values for the traversing velocity the controller has enough time to correct the position error during the traversing phase. Subsequently, the same analysis is performed on the robot simulator (shown in figure 6-8) on the same experiment. However, in the simulation the height of the error is similar to the one from the real test, but the duration of the error is increased. The cause of this phenomenon could be from the fact of the non-reality of the simulation.

The angular trajectory error caused in the simulation by the movement of the hip joint with different traversing velocities is shown in figure 6-9. The error acquired from the simulation shows similarity to real robot test. This circumstance shows the capability of the simulator to anticipate the system's behavior caused in the reality.

6.2. Multi-joint test

In this test, one leg of the robot is evaluated. Each leg consists from seven degree of freedom (i.e., each leg has one transversal, two frontal and tree lateral joints plus the toes joints); although in this test the evaluation is performed for two of the lateral joints (i.e., hip lateral and Knee joint). The evaluation is done by comparing the traversed trajectory of the robot's leg with the trajectory command. The comparison is also prepared for the actual robot's leg and the simulation of the leg.

Capturing the traversed trajectory of the robot's leg is provided using Image Processing technique. In this technique, a high frame rate camera (BASLER A302bc, 60 frames per second) is used to track some patches which are glued to the robot. The patches used for this technique are shown in figure 6-10.

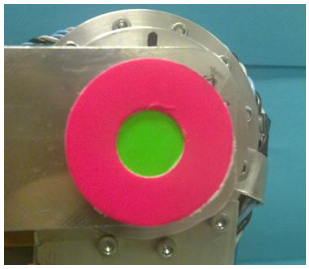


Figure 6-10: The patch used for tracking the traversed trajectory

The camera is placed in a lateral distance of one meter from the robot to track the patches glued to the robot's leg (as it is shown in figure 6-11). Using the video stream taken from the leg during the test by the camera and the image processing software (written in Visual Studio 2008 and is shown in Appendix A), the trajectory of the robot is extracted (as it is shown in Figure 6-13).

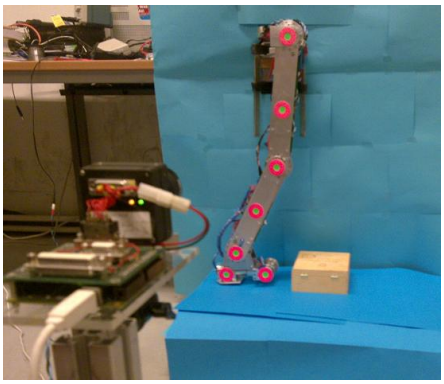


Figure 6-11: Camera placed in one meter lateral distance of the robot's leg for patch tracking

To distinguish between the patches, the software uses a low pass filter on the position of the patches. By using this method, the patches are tracked and distinguished from each other.

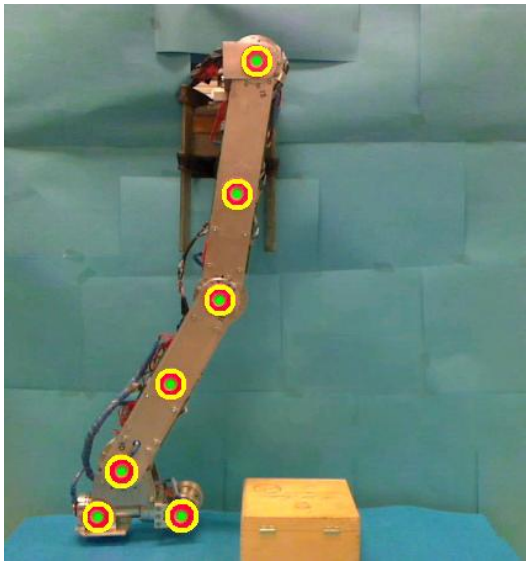


Figure 6-12: Patch detection result used for finding the traversed trajectory

Figure 6-13 shows the extracted trajectory of the patches during a half gait. The traversed trajectory is stored in a spreadsheet [17] which can be used in Microsoft Excel or in Matlab for further analysis.

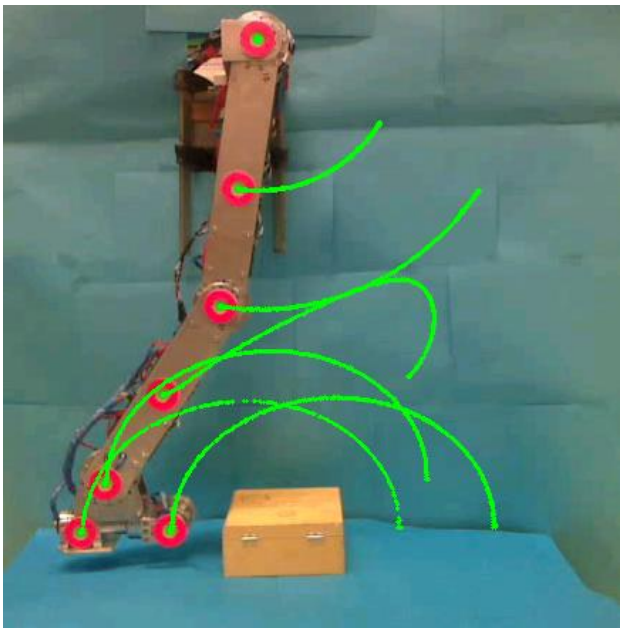


Figure 6-13: Traversed trajectories resulted from image processing patch detection

[17] - Spreadsheet is a computer data format which consists from rows and columns of data cells.

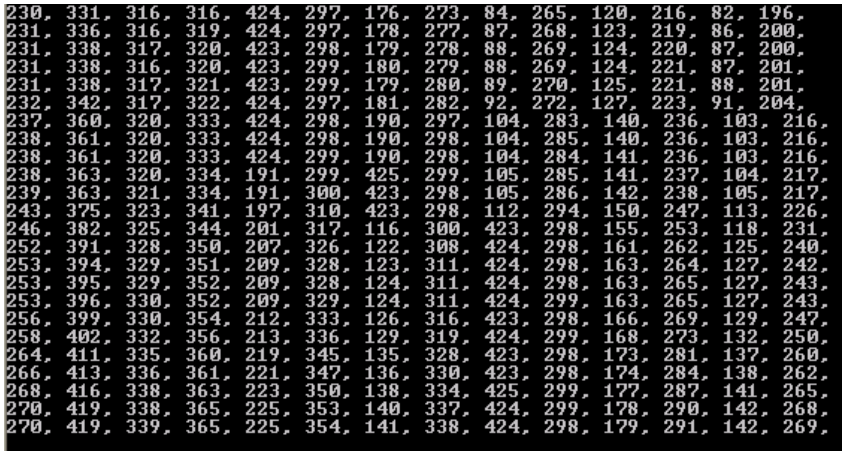


Figure 6-14: Detected coordinates from the patches tracking row data (consists from X and Y, repeatedly)

In this level of the test (evaluating a single leg), the simulator of the robot is reduced to a single leg. In this simulation, the hip of the robot is attached to the root weld of the simulation environment. The trajectory command is given to the joints of the leg. By using the Body Sensor (a component from Matlab Simulink SimMechanics) the traversed trajectory of the leg's ankle is extracted. The extracted trajectory is used for comparison with the real robot.

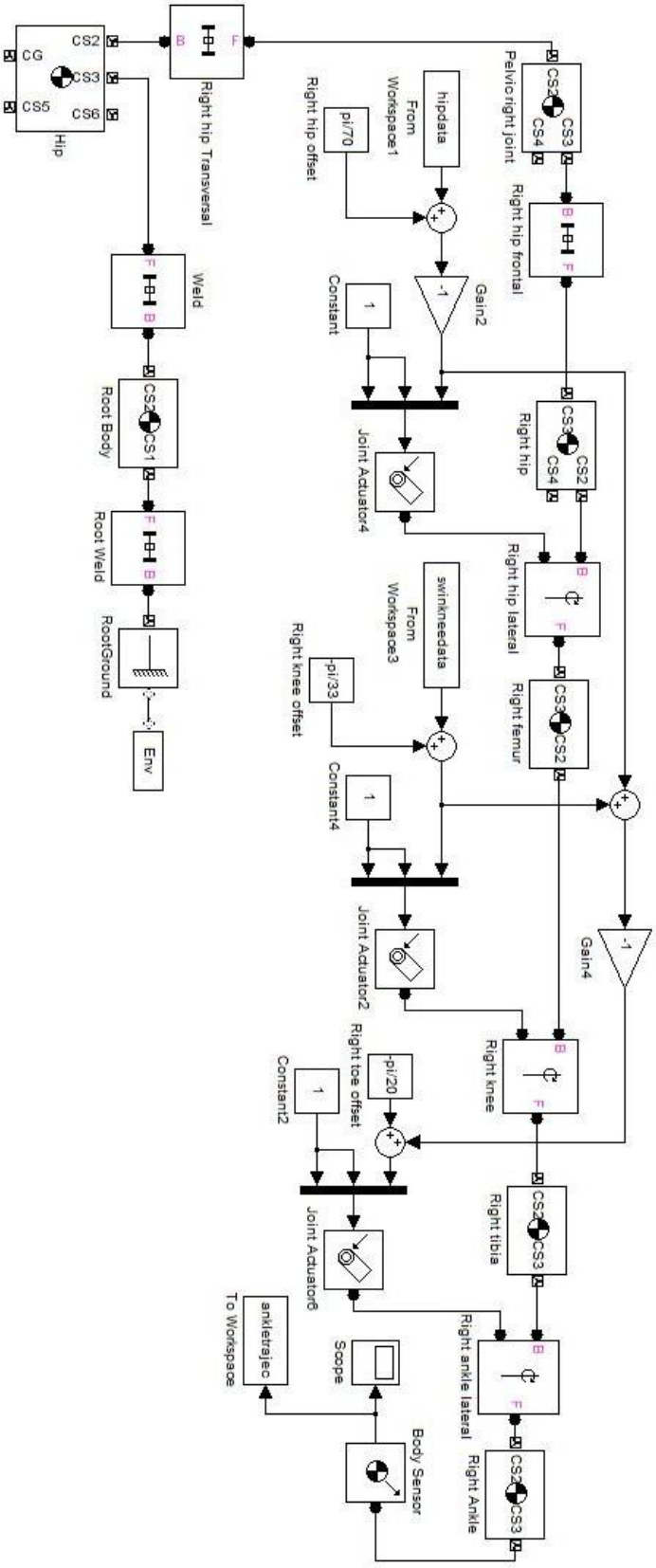


Figure 6-15: Simulink diagram of Archie's swinging leg

Figure 6-16 shows the reduced simulation for the single leg.

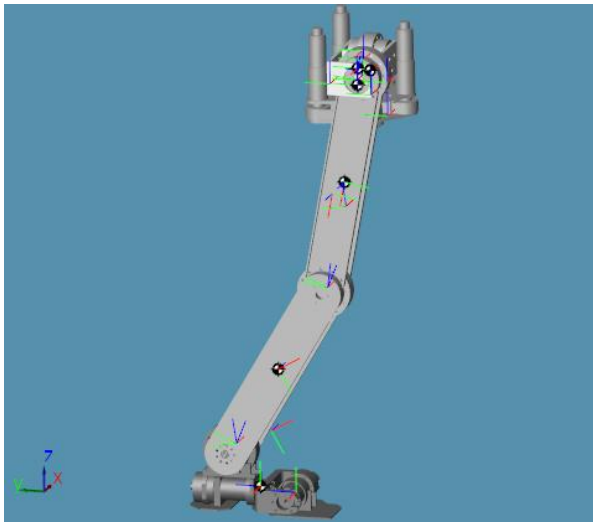


Figure 6-16: Single leg simulation

The test for the leg is applied for a half gait, full gait and a trapezoidal movement.

6.2.1. Half gait test

In this test the robot's leg is evaluated for a half gait (swinging phase) movement. The trajectory commands (the joints trajectory command) are calculated by the central controller using the inverse kinematics model of the robot (leg). Figure 6-17 shows the traversed trajectory of the swinging leg's ankle during the test.

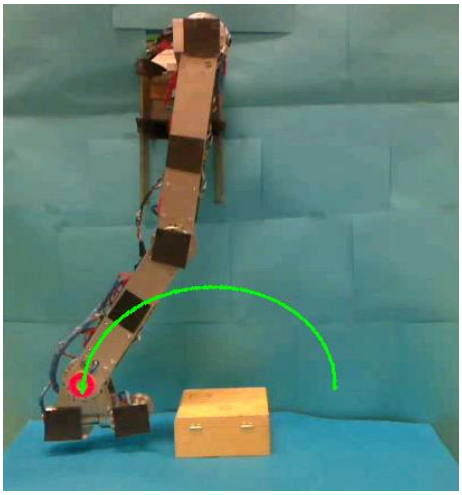


Figure 6-17: Swinging leg's ankle traversed trajectory from lateral view

Figures 6-18 and 6-19 show the traversed trajectory vs. the trajectory command and the position error during the movement, respectively for the half gait (swinging leg) test.

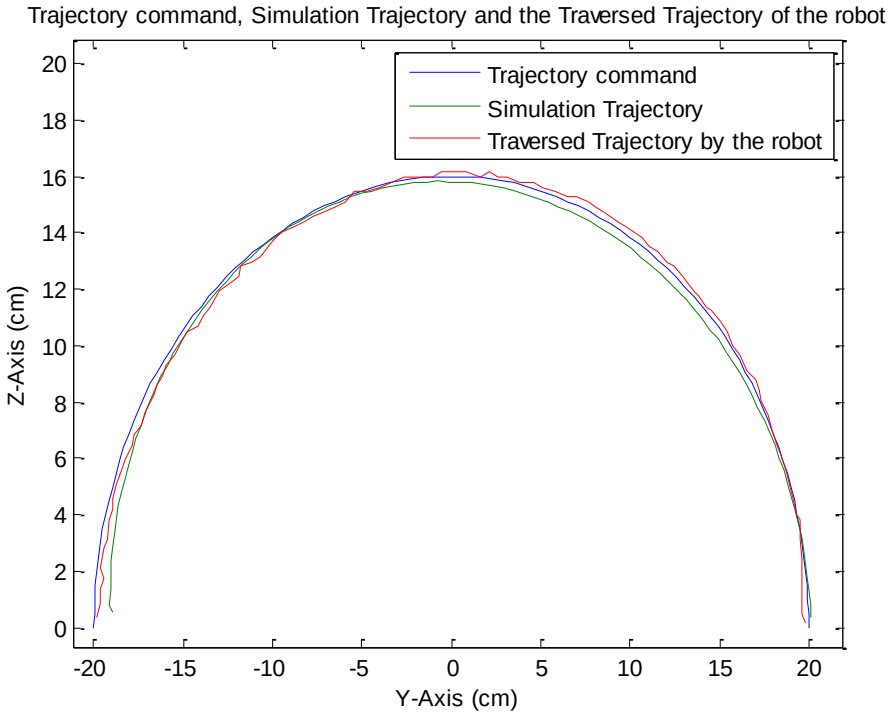


Figure 6-18: Traversed trajectory by the real robot, simulation vs. the trajectory command for the half gait

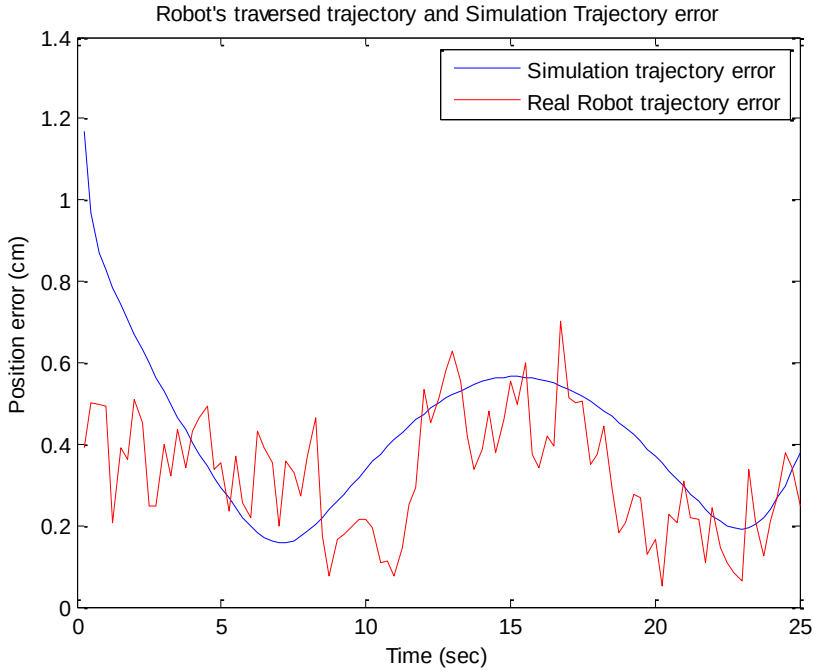


Figure 6-19: Position error of the traversed trajectory by the real robot and the simulation for half gait

The error obtained from the half gait test on the real robot is limited under 0.6cm, which is an acceptable value. In addition, the error escalation in around the 15th second (in graph 6-19) is also predicted by the simulator. The main reason for the occurred error is the changes of the load specification (i.e., the gravitational force changes the direction because the leg is not climbing anymore and the end effector movement direction is facing downward). The following error results from the hip joint. The hip joint (as it is shown in figure 6-20) gets relatively higher error around the 15th second (figure 6-21) because the main effect from the gravitational force is applied to the hip joint.

In the simulation results, it shows the same effect in the error around the 15th second, which shows that the gravity effect is properly implemented to the simulator.

The angle trajectory for the knee joint, the error resulted from the simulation and the real robot, the velocity command versus the joint velocity in the simulation and the robot are shown in figures 6-20, 6-21 and 6-22 respectively.

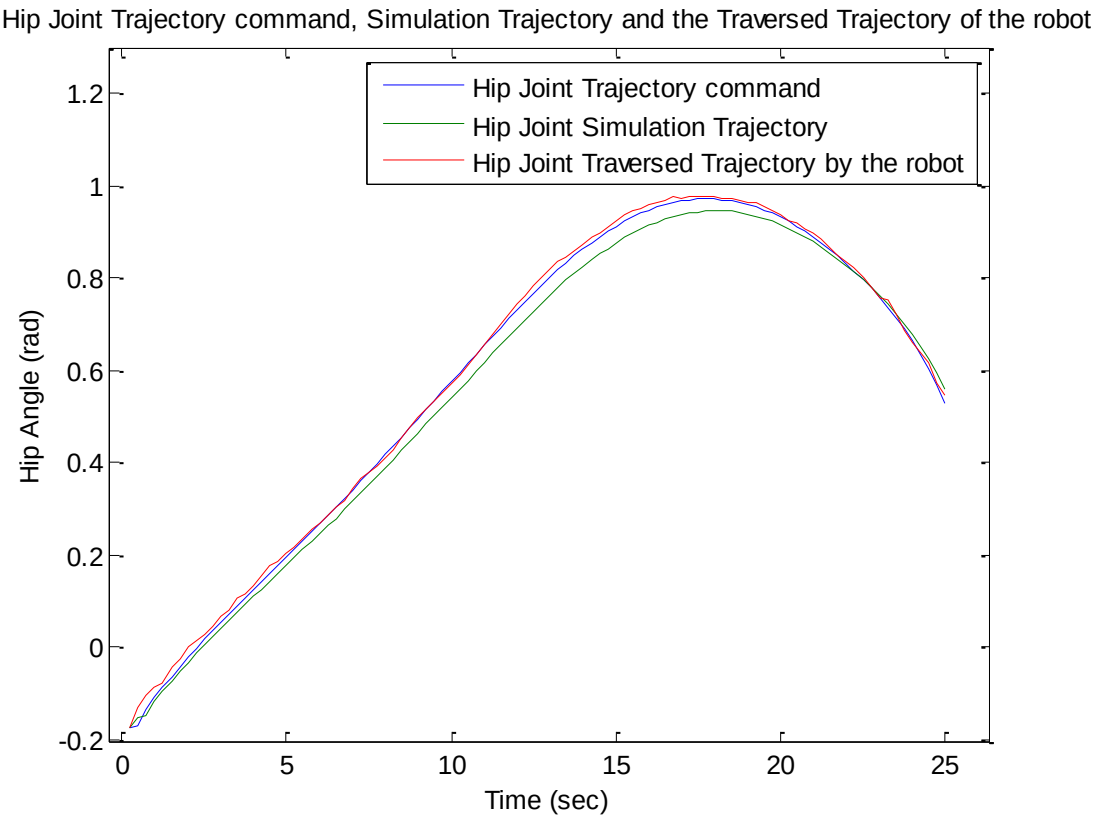


Figure 6-20: Hip joint angle trajectory command, simulation and the real robot for half gait test

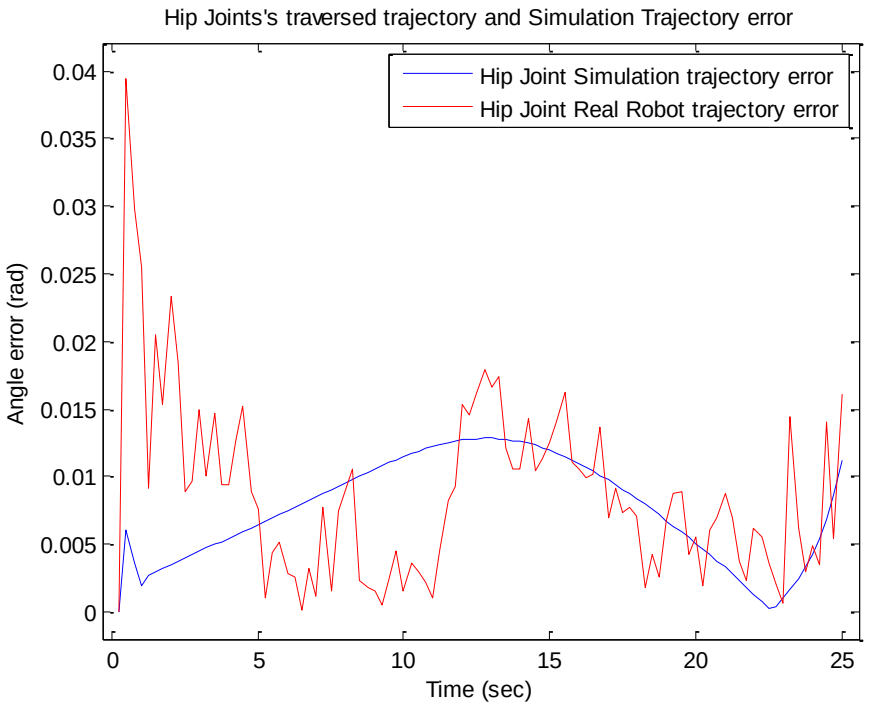


Figure 6-21: Angle error resulted from simulation and real robot in the hip joint during the half gait test

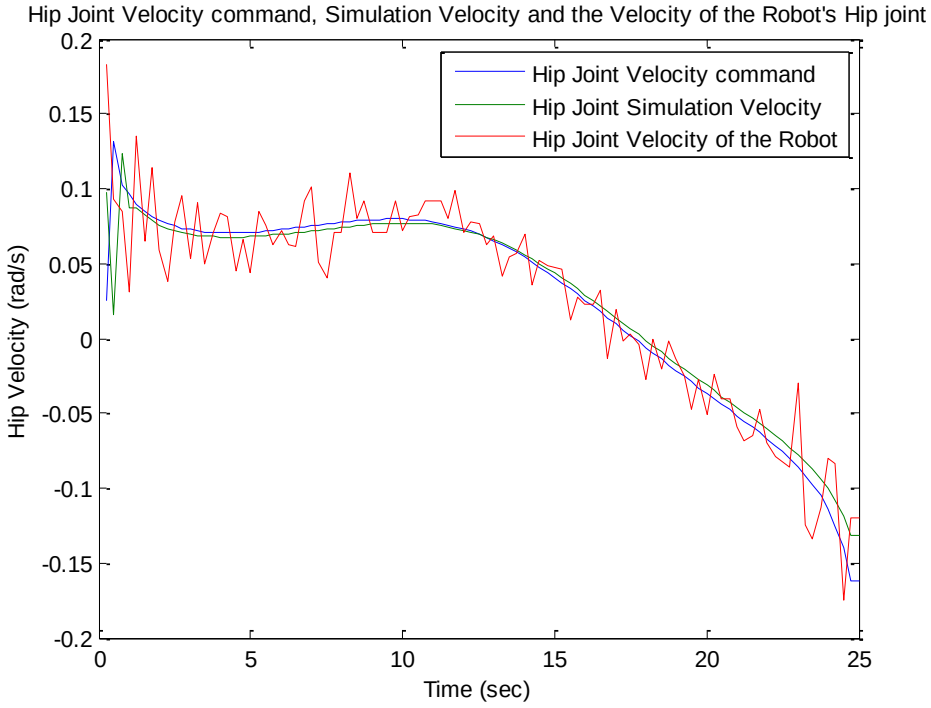


Figure 6-22: The velocity command compared with the simulation and real robot of the hip joint during the half gait test

The angle trajectory for the knee joint, the error resulted from the simulation and the real robot are shown in figures 6-23 and 6-24 respectively.

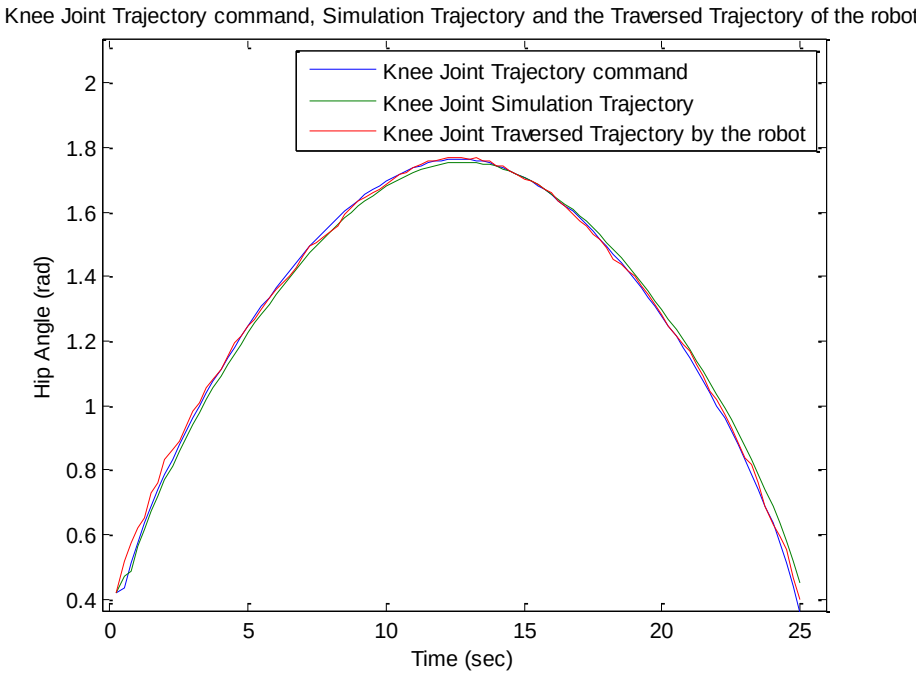


Figure 6-23: Knee angle trajectory command vs. the simulation and the real robot trajectory for half gait

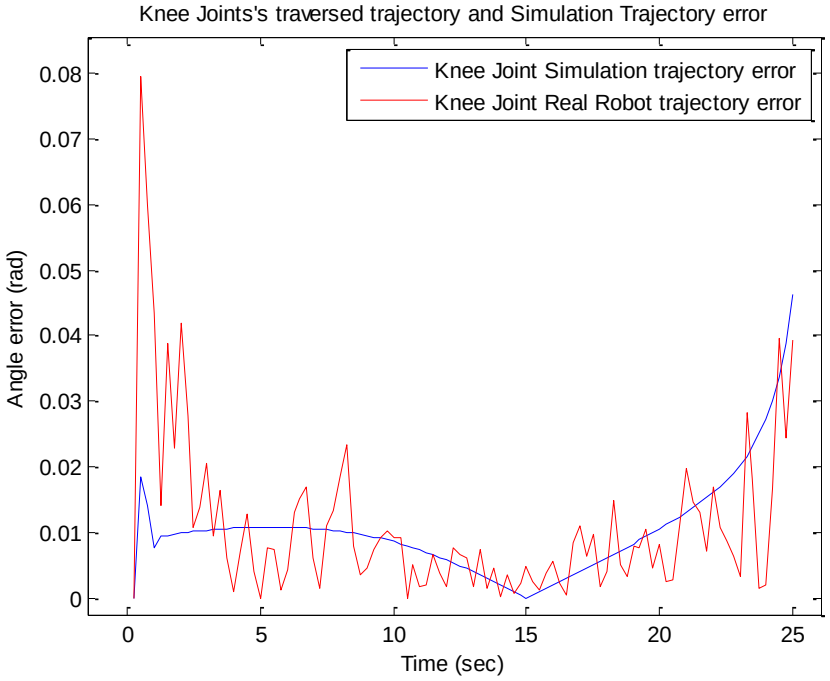


Figure 6-24: Angle error resulted from simulation and real robot in the knee joint during the half gait test

The knee joint shows fairly moderate error rate compared with the hip joint, the reason might be the lower mass of the tibia (i.e., the tibia is lighter than the whole leg). In the beginning of the motion, the error is in a higher range which is caused because the motion controller is overcoming the static inertia of the tibia. In addition, the error rises again because of the end effector movement direction that is facing downward.

The angular velocity command for the knee joint versus its angular velocity in the simulation is shown in figure 6-25. The real robot shows more instability in the velocity compared to the simulation. However, the velocity variation in the real robot is in an acceptable range.

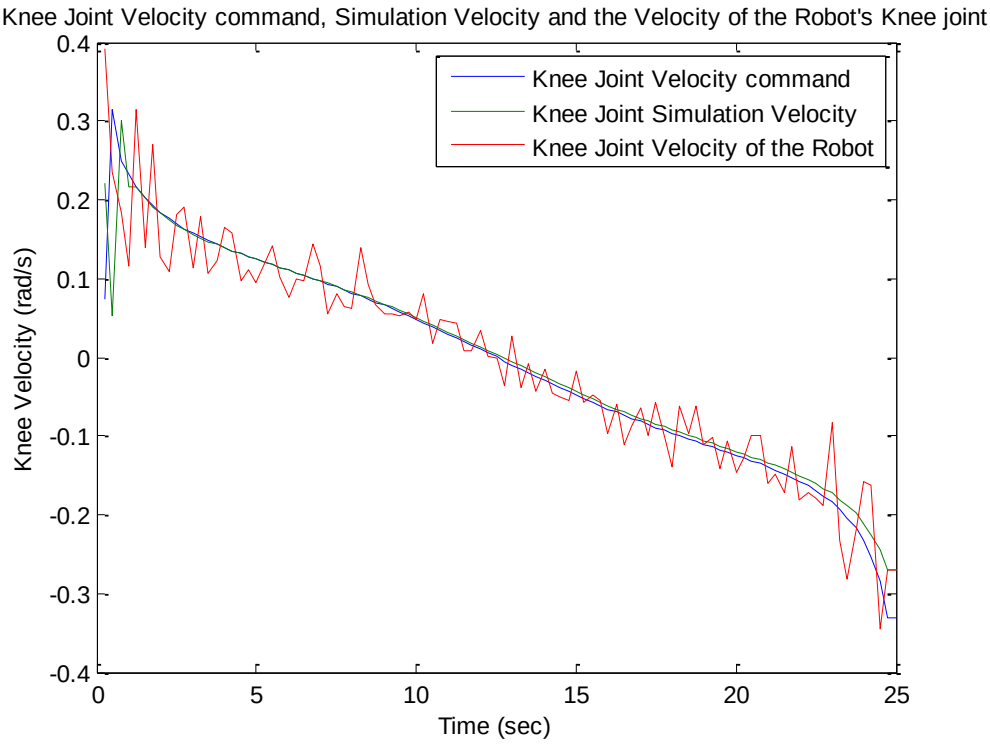


Figure 6-25: The angular velocity command compared with the simulation and real robot of the knee joint during half gait test

6.2.2. Full gait test

The full gait test is an extended version of the half gait (swinging) test. The full gate test includes the supporting phase. The full gait movement moves the robot one step further. Figure 6-26 shows the right leg of the robot and the traversed trajectory for a full gait movement. In figure 6-27 the traversed trajectory of the robot, the traversed trajectory in the simulation and the trajectory command are shown for comparison.

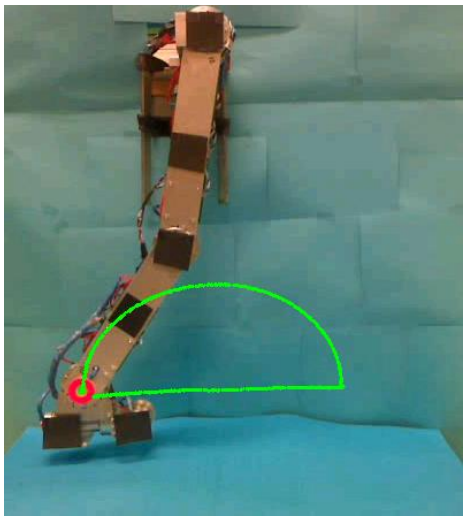


Figure 6-26: Full gait traversed trajectory resulted from patch tracking

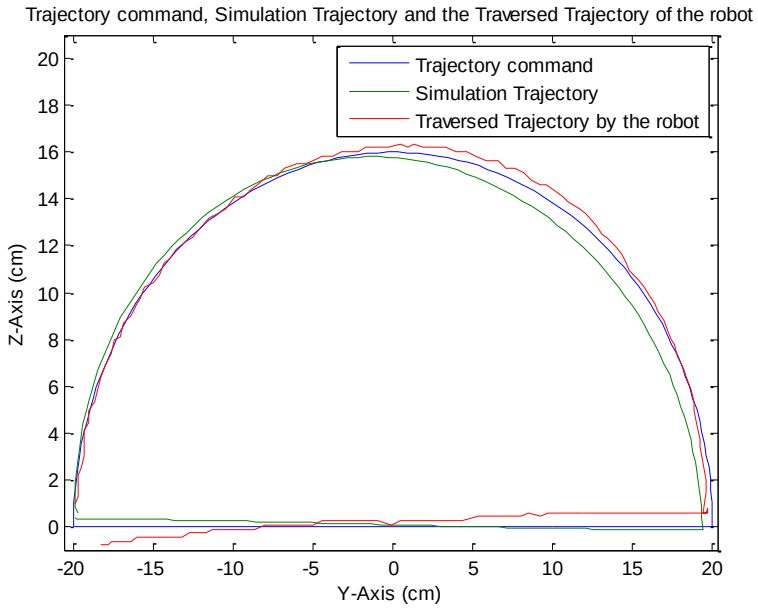


Figure 6-27: Trajectory command, traversed trajectory of the robot and the simulator for the full gait

The resulted error from the real robot and the simulation of the robot are shown in figure 6-28.

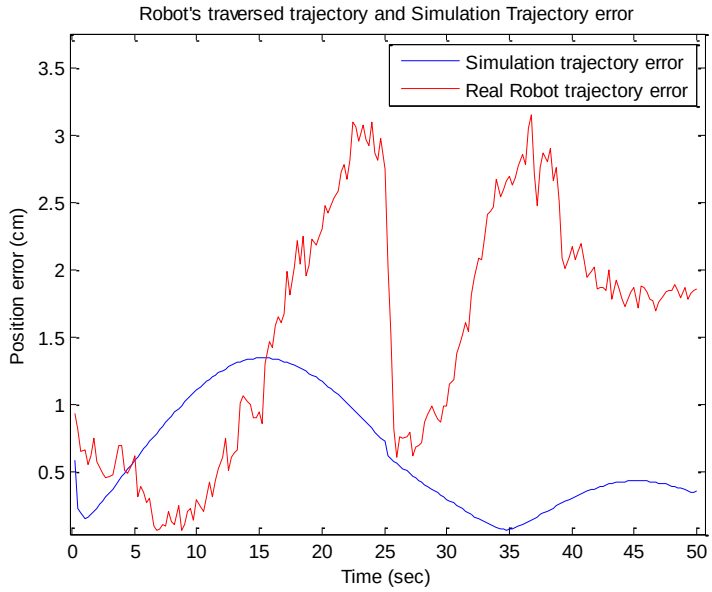


Figure 6-28: Position error of the robot's traversed trajectory and the simulator trajectory of the full gait

The position error obtained from the real robot around the 35th second increased significantly; however, the simulation error decreased to a minimum level during the test. The reason for the following phenomenon could be from the coincidence of the center of mass of the robot with the direction of the robot's leg which causes the maximum gravitational force on the robot's joints. The angle trajectory for the hip joint is compared with the angle trajectory of the simulation and the real robot (figure 6-29).

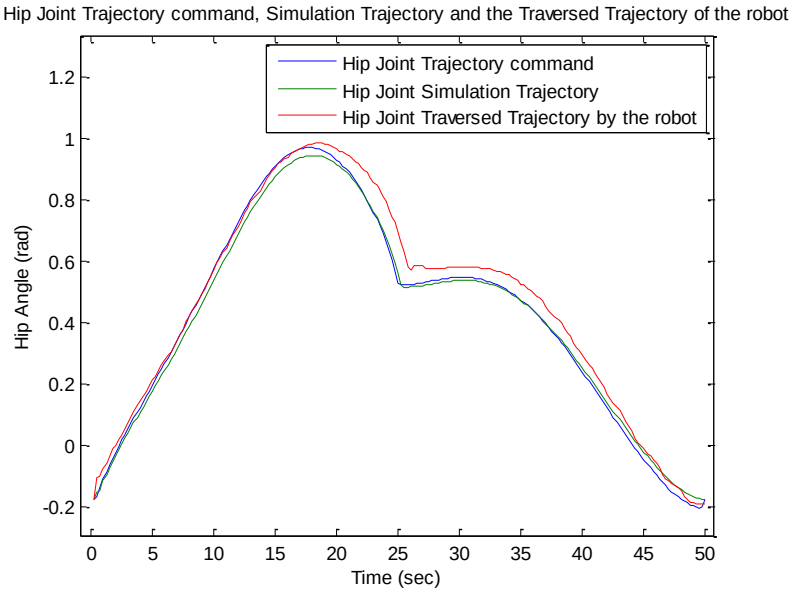


Figure 6-29: Hip joint angle command vs. the simulation and the real robot for the full gait movement

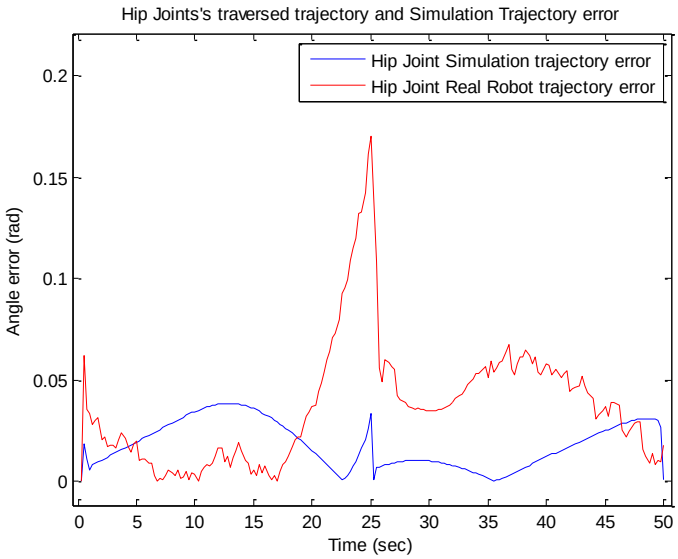


Figure 6-30: Angle error resulted from simulation and real robot in the hip joint during the full gait test

The error that results from the real robot's hip joint and its simulation during the full gait is shown in figure 6-30. The hip joint error increases around the 25th second which is where the control system changes the motion phase from swinging to supporting. Similar to the half gait test the error increases in the beginning and in the end of the swinging gait (which comes from the coincidence of the gravitational force and end effector movement direction).

The angular velocity command of the hip joint is compared to the velocity of the real test and the simulation of it (figure 6-31).

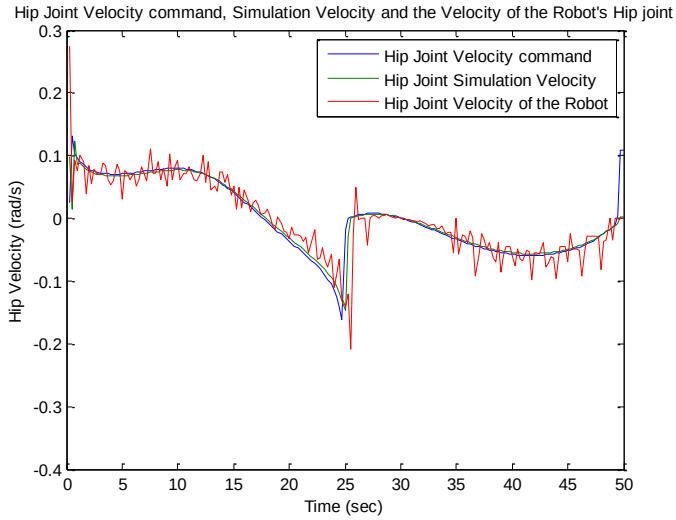


Figure 6-31: The angular velocity command compared with the simulation and real robot of the hip joint during the full gait test

The angle trajectory command for the knee in the full gait movement, beside the trajectory resulted from the real robot and the simulation are shown in figure 6-32.

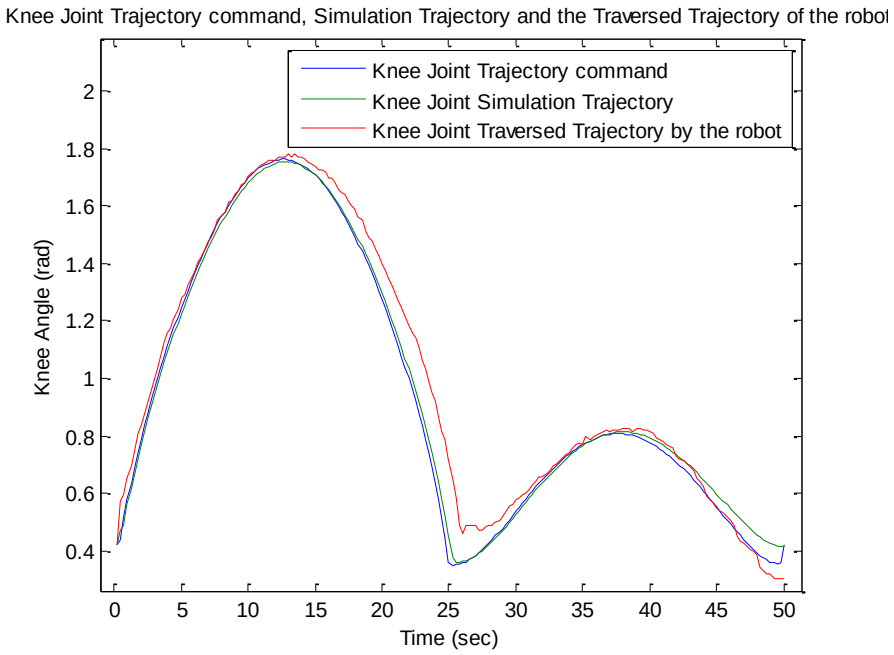


Figure 6-32: Knee joint angle trajectory command, simulation and the real robot for the full gait

The resultant angle error from the real robot and the simulation are shown in figure 6-33.

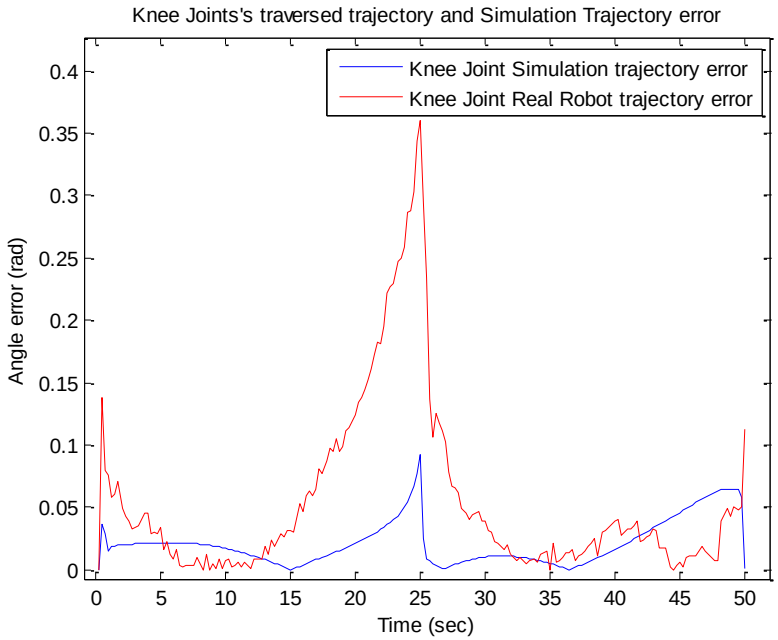


Figure 6-33: Angle error resulted from simulation and real robot in the knee joint during the full gait test

For the knee joint, the same condition is happening (figure 6-32) during the changing from the swinging phase to the support phases the error increases swiftly. During the support phase, the error rate is fairly steady and that happens because of the constant height of the robot's leg (i.e., the gravitational force is constant for the leg).

The angular velocity applied to the knee joint during the full gait test is compared to the angular velocity of the real robot and its simulation which is shown in figure 6-34.

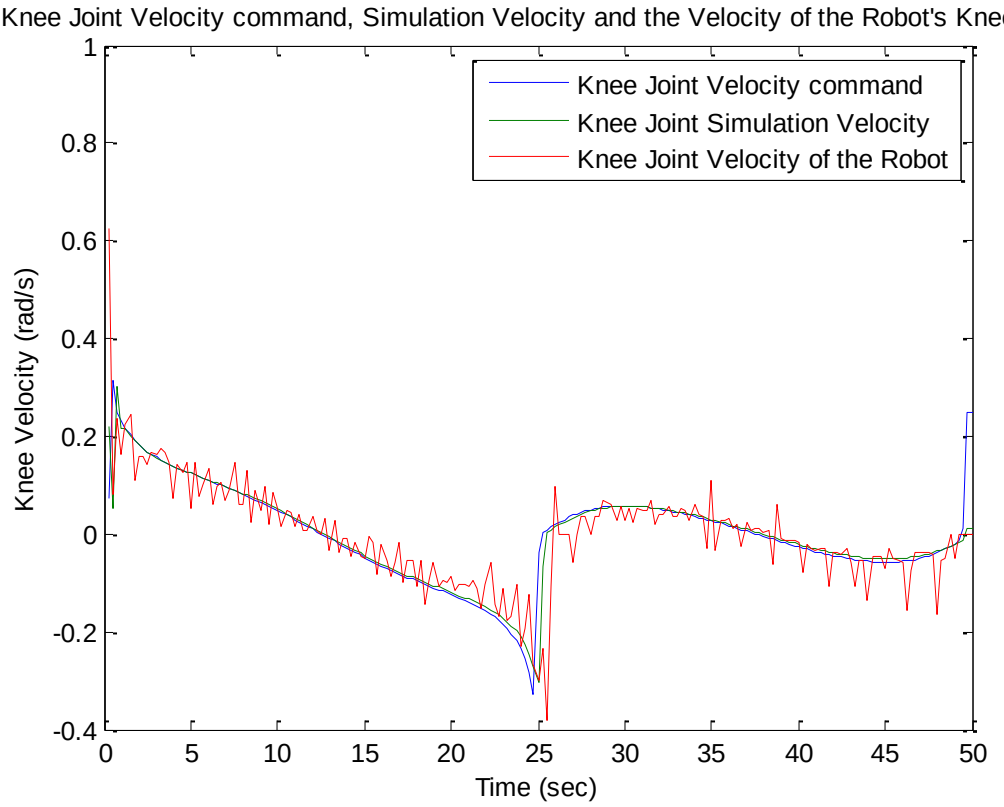


Figure 6-34: The angular velocity command compared with the simulation and real robot of the hip joint during the full gait test

The velocity trajectory of the real robot (shown in 6-34), shows a good response to the velocity command, despite the instability which is an acceptable range for the real robot test.

6.2.3. Trapezoidal test

Using a trapezoidal trajectory in the gait imitation of the robot causes vibration and instability. The trapezoidal trajectory is not used in the robot for walking (the elliptical movement is preferred to it); however, it is used for the multi joint test. The evaluation is made by comparing the simulation and the real robot (as the same for the elliptical trajectory tests). Using a trapezoidal trajectory can be useful for an accurate evaluation of the control system and the system response. For instance, the response to the edged corners and the flat part of the trajectory are showing the ability of the control system and the joint controllers.
In figure 6-35 the traversed trajectory of the Ankle joint of the robot is shown during the trapezoidal test.

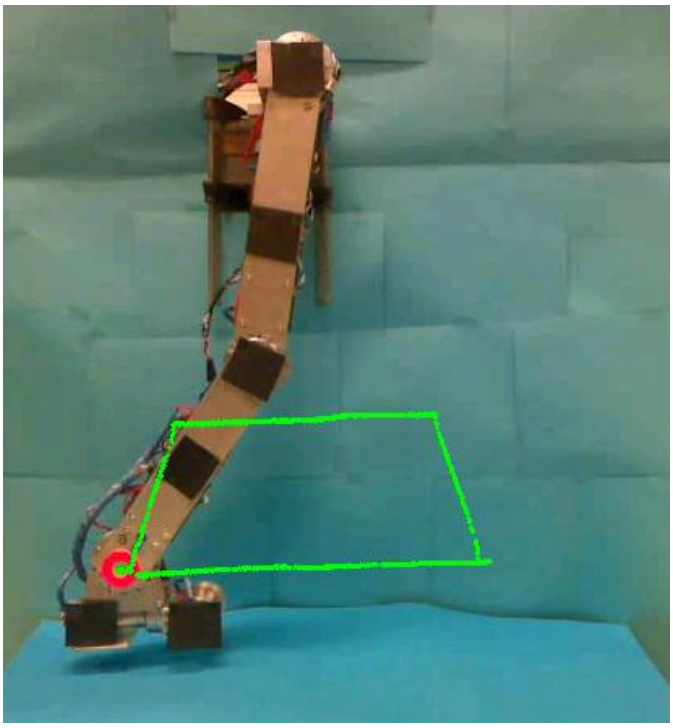


Figure 6-35: Trapezoidal traversed trajectory resulted from patch tracking

The comparison of the traversed trajectory in the real robot and the simulator with the trajectory command is shown in figure 6-36.

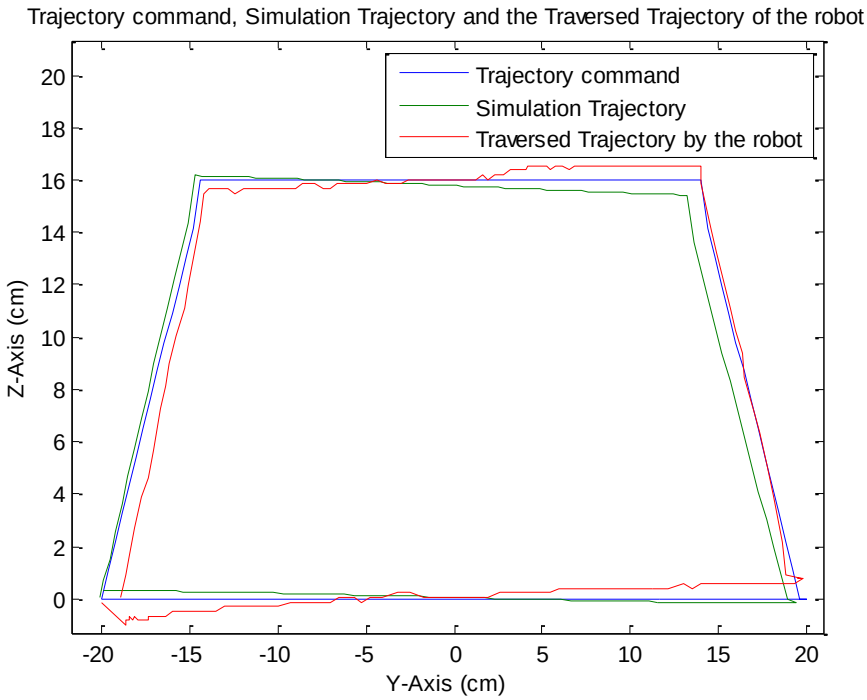


Figure 6-36: Trajectory command, traversed trajectory of the robot and the simulator for the trapezoidal

The error comparison between the traversed trajectory of the real robot and the simulation is shown in figure 6-37.

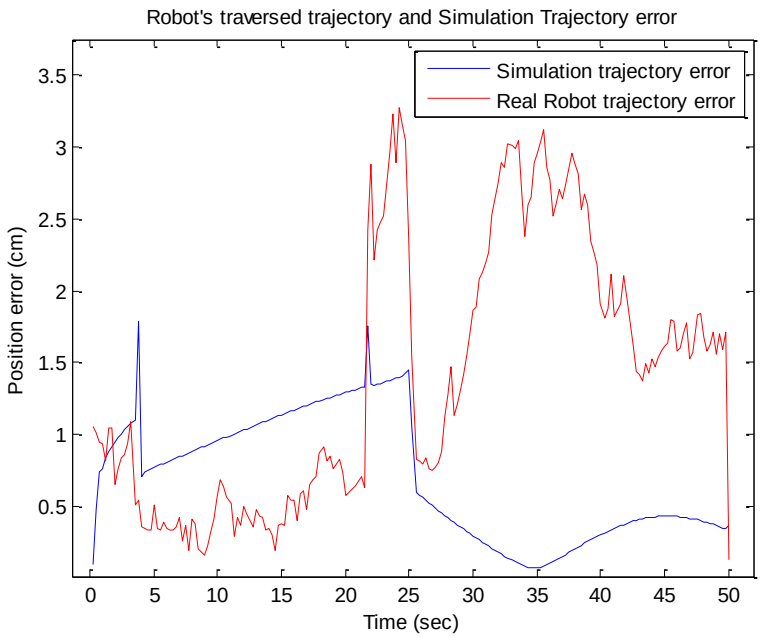


Figure 6-37: Position error of the robot traversed trajectory and the simulator trajectory by trapezoidal test

The position error from the end effector during the trapezoidal motion test shows two spikes around the 5th and the 20th second. The main reason for these error spikes is the high edge of the trajectory of the command (during switching the direction from vertical to horizontal and vice versa, in the trapezoidal trajectory motion). Moreover, the error increases in a fairly high rate during the vertical movement which comes from the gravitational effect that coincides with the end effectors motion direction. In addition, the error increased significantly around the 35th second which could be caused by the gravitational force which also occurs in the full gait test at the same time.

The angle of the hip joints for the trapezoidal test is compared with the simulation result and the data obtained from the real robot which is shown in the figure 6-38. Also, the error occurred from the real robot and the simulation of the robot is shown in figure 6-39. The angle trajectory of the hip shows two relative high edges in the 20th and the 25th second (as well as in the error comparison graph shown in figure 6-39). During the 20th second that was also explained before, the high edge of the trajectory is the main cause. The error increases around the 25th second because of the phase change (i.e., phase exchange from swinging to support phase).

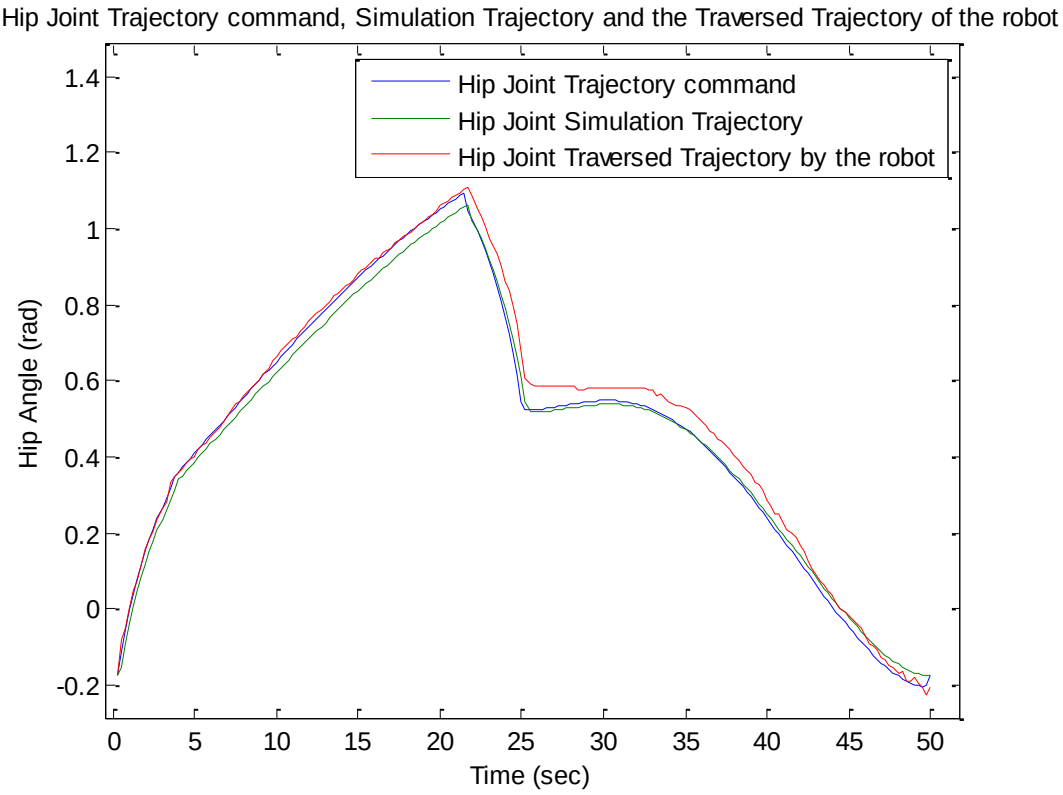


Figure 6-38: Hip joint angle trajectory command, simulation and the real robot for the trapezoidal test

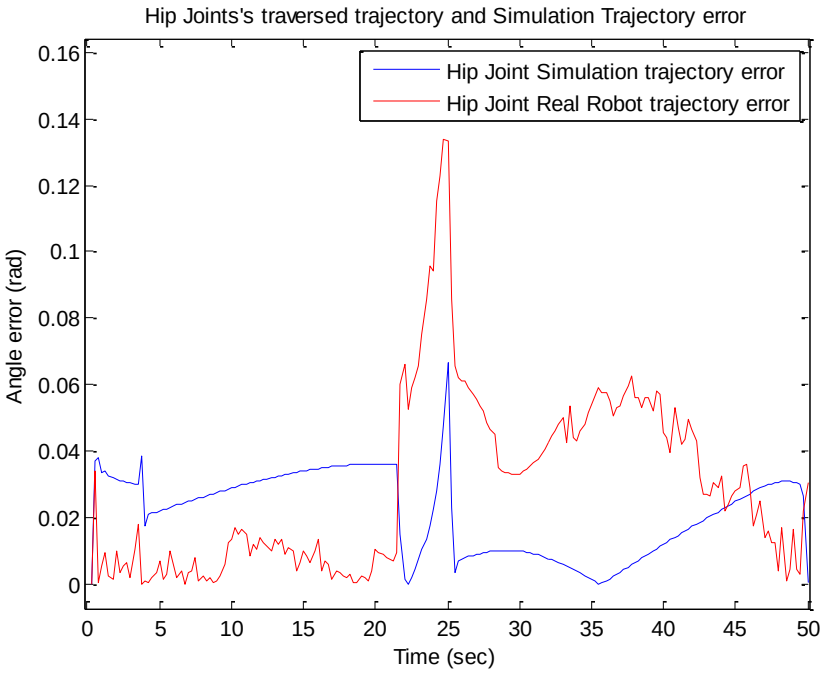


Figure 6-39: Angle error resulted from simulation and real robot in the hip joint during the trapezoidal test

The Velocity command applied to the hip joint is compared with the velocity of the real robot's joint and the simulation in figure 6-40.

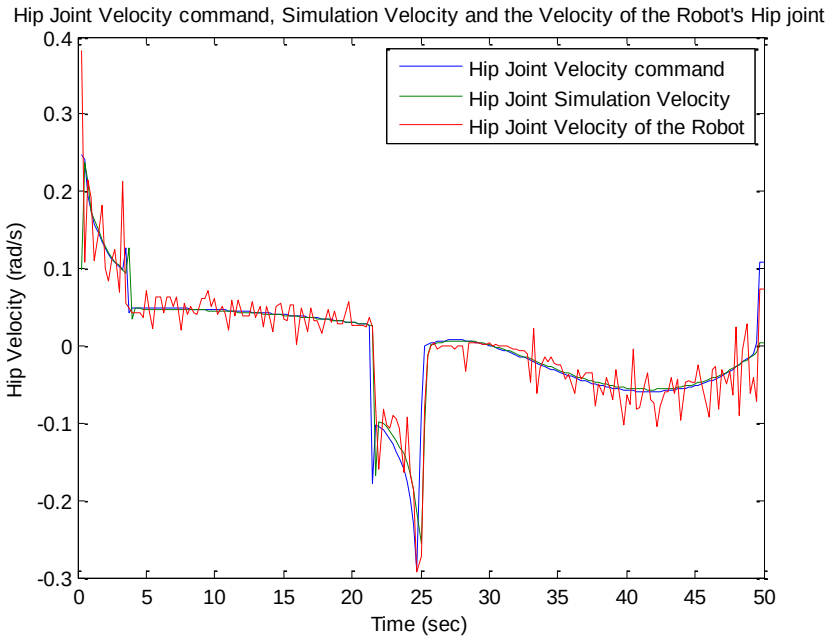


Figure 6-40: The angular velocity command compared with the simulation and real robot of the hip joint during the trapezoidal test

In figure 6-41, the knee joint's trajectory command is compared with the traversed trajectory of the real robot and the simulation.

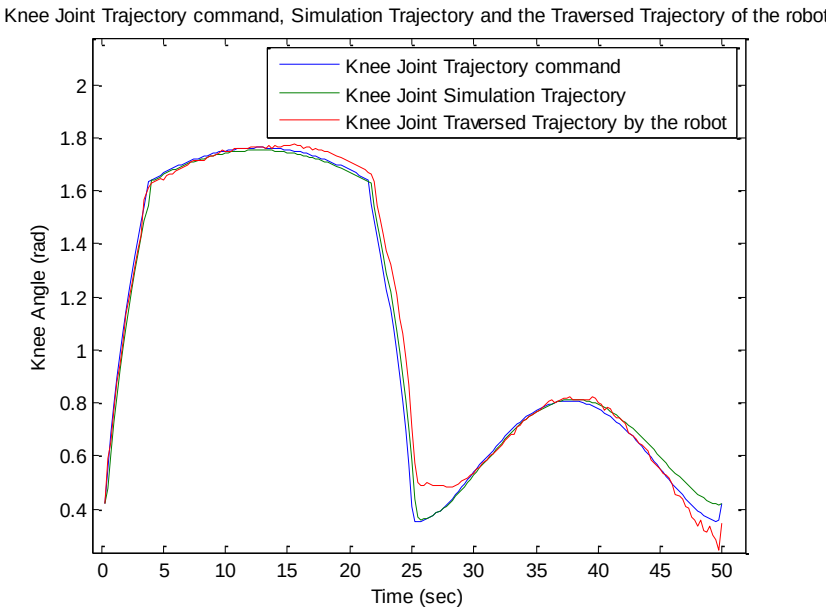


Figure 6-41: Knee angle trajectory command vs. the simulation and the real robot trajectory for the trapezoidal test

The error caused by the real robot and the simulator is shown in figure 6-42.

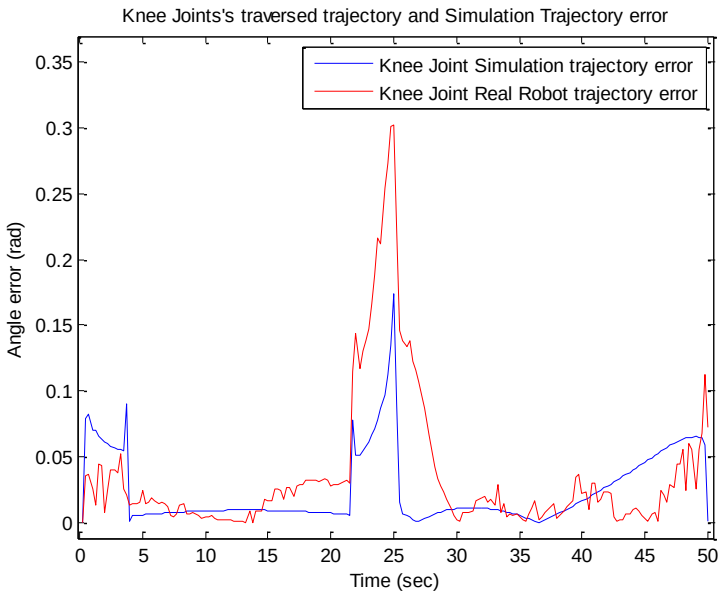


Figure 6-42: Angle error resulted from simulation and real robot in the knee joint during the trapezoidal test

The knee joint shows similar error spikes (regarding to figures 6-41 and 6-42) which are also caused by the high edges (in 5th and 20th second) from the trajectory command. Also, the error during switching the phases (i.e., switching from swinging to support phase in the 25th second) is shown in the error diagram.

The velocity command of the knee joint is compared with the velocity of the real robot and the simulator and is shown in figure 6-43.

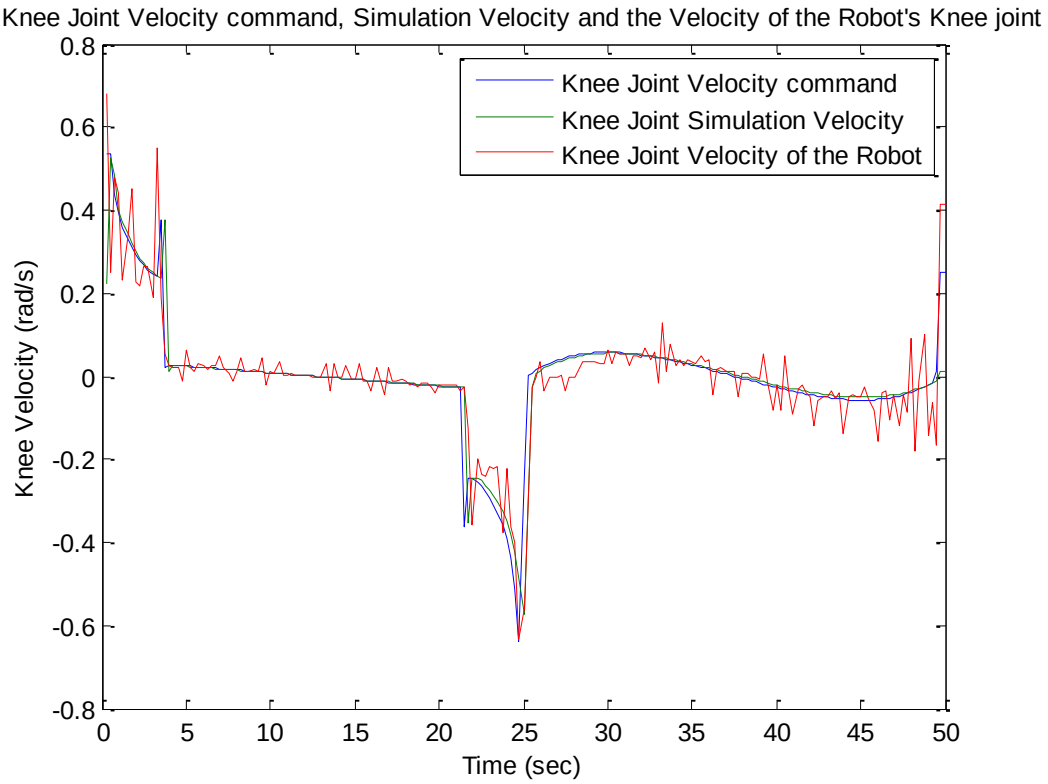


Figure 6-43: The angular velocity command, compared with the simulation and real robot of the knee joint during the trapezoidal test

6.3. Walking test

In this test, the walking sequence of the robot is provided from three static poses. The poses of the robot are, crouch center, crouch left and crouch right. Running sequentially the crouch left and the crouch right and putting the center crouch in between causes a basic dynamic movement (walking) in the robot. After executing of each movement, the robot should delay a minimum time to stabilize and get the overall stationary pose of the robot. The real test shows the delay time with 200ms minimum (i.e., the delay time should be supposed more than 200ms in order of robot's stability).

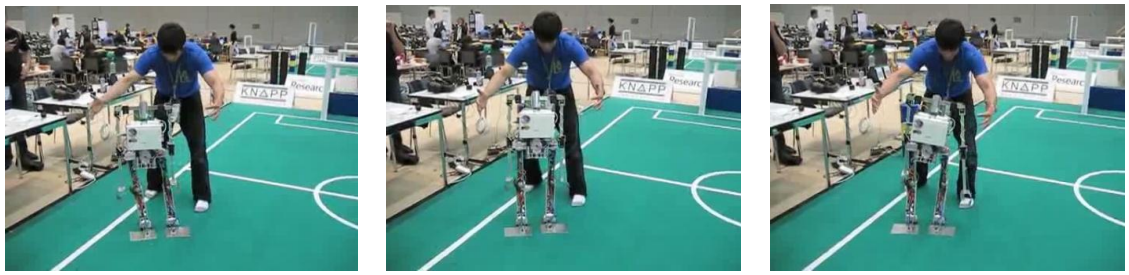
Table 6-1 shows the joint angle values for the three poses that the robot control system should execute for the basic dynamic walk. The values shown in the table 6-1 are only the desired angles. The control system generates more detailed angles values by using interpolation (i.e., the angle trajectories for the joints are calculated by the control system of robot using interpolation).

The angles of the joints are set by using table 6-2 (angles are relative to T-form):

Joints	Crouch left		Crouch center		Crouch right	
	Left	Right	Left	Right	Left	Right
Hip transversal	0	0	0	0	0	0
Hip frontal	5	-5	0	0	-5	5
Hip lateral	30	30	25	25	30	30
Knee lateral	10	25	25	25	25	10
Ankle lateral	0	15	15	15	15	0
Ankle frontal	-5	5	0	0	5	-5
Toe lateral	0	0	0	0	0	0
Backbone frontal	-5		0		5	
Backbone transversal	0		0		0	

Table 6-1: Situation of the angles of individual joints for dynamic walk

The crouch steps are shown in figure 6-44:



(a) **(b)** **(c)**

Figure 6-44: Basic dynamic walk: (a) crouch left, (b) crouch center, (c) crouch right

This type of walking requires more time for the robot to move from one point to another point, in comparison with human gait imitation movement. The basic dynamic walk movement is easy to implement, and it is also useful to evaluate the control system performance of the robot.

The more advanced test in the robot is the human gait imitation. In this test, the robot tries to imitate the human gait based walking. For this imitation the robot control system uses two movement phases, one for left step and one for the right step. The positions between these two steps are calculated using the interpolation. The desired angles are shown in table 6-2.

Joints	Step left		Step right	
	Left	Right	Left	Right
Hip transversal	5	5	-5	-5
Hip frontal	5	-20	-20	5
Hip lateral	20	-5	-5	20
Knee lateral	20	10	10	20
Ankle lateral	10	15	15	15
Ankle frontal	10	-10	-10	10
Toe lateral	0	0	0	0
Backbone frontal	-5		5	
Backbone transversal	5		-5	

Table 6-2: Situation of angles of individual joints for human like gait

Figure 6-45 shows the two movement phases for the robot during human gait imitation.

(a) (b)

Figure 6-45: Human gait imitation: (a) Step right, (b) Step left

The walking motion (Human gait imitation) that is shown in figure 6-45 is fairly acceptable, despite the fact it needs more development and improvement in further works. Although a real human walking trajectory can be obtained by using the presented technique in this thesis (i.e., using image processing technique) and applied to the robot in order to have a more natural and stable human walking imitation. Further improvement suggestions are presented in the next chapter.

Chapter 7

7.Summary and Outlook

In this thesis, a new control system has been designed for a humanoid robot. The control system is split in individual joint controllers that communicate with the central controller trough a data network. Using an embedded system, the central controller is built on a system on chip (i.e., the system is implemented on a Virtex 4 FPGA from Xilinx which contains an embedded PowerPC hardware core) platform. In mobile robots, the ratio between the energy consumption of the robot (Robot's wattage) and the energy resource capacity is an important factor. Increasing the energy resource capacity (batteries) causes more weight on the robot which will also increase the energy consumption of the robot. Decreasing the energy consumption in the robot can be performed by using embedded systems. By using embedded systems (e.g., SOC) in the robot, the necessary energy for data processing that a general purpose computer provides (i.e., personal computer) can be afforded by lowering the energy consumption. In addition, the reliability of the system is increased by using embedded systems instead of general purpose computers.

Using the simulator for the robot is beneficial from different aspects. For example, the simulator eases the development and the motion analysis for the robot by creating a virtual robot. Many of the physical parameters (e.g., position, velocity and acceleration, total center of mass and moment of inertia) of the robot can be monitored during the movement of the robot.

Comparing the results from the real robot with the simulation of the robot can be a way to distinguish the system failures from the algorithm failures. For instance, an error caused from the inverse kinematics model in the robot can cause error in the real robot's movement trajectories. By using a simulation the same error will be shown. Then, by comparing the results and the error from the real robot and the simulation of the robot the error source can be differentiated. This circumstance will accelerate the development and improvement of the robot and the control algorithms used on it.

7.1. Future work

Since the developed control system for the robot is a new work, some improvements can be applied to achieve a better performance. In this section some of these improvements and suggestions are presented for the future work on the control system of the robot. The suggestions for the improvements of the control system are presented based on the units that they should be applied (i.e., the suggested improvements for each unit are presented in a certain part).

7.1.1. Central controller improvements

This section talks about the proposed suggestions to improve the performance of the central controller of the robot:

- The central controller is a busy unit in the robot. Since all the data from the units are collected and processed in this unit, the load of process is high. Therefore, any improvement in this unit is critical and remarkable.
- The Data Acquisition Unit (DAU) takes a major load off from the main processor (described in chapter 5). This unit can be extended to interpret and combine the data received from the joint controllers. The extension could be based on hardware to reduce the processing load on the central processor.
- Repeatedly operation like “the robot’s total center of mass” calculation can be implemented on the hardware
- The robot motion planner is based on off-line Zero Moment Point (ZMP) (i.e., all the motions are preprocessed by the simulation of the robot, and are later applied to the robot statically). The robot cannot operate in non-ideal environments such as non-even terrains. To achieve this ability in the robot, using pressure sensors in the sole of the robot can be useful.
- More advanced dynamic walk can be developed on the robot using the feedback data from the Inertial Measurement Unit (IMU), based on the “moment of inertia” of the “center of mass” of the robot.
- The walking trajectory can be switched to the natural human walking trajectory. By using a natural human walking trajectory, the robot can walk naturally like a human.

7.1.2. Joint controller improvements

In this section, the proposed suggestions to improve the joint controller of the robot are presented:

- The joint controllers in the robot are working with constant controller compensator parameters (PID gain values). These values are extracted from tuning the joint controller. Since the load specification reflected to the joints is related to the overall pose of the robot (discussed in chapter 4); the tuning of the joint controller is very difficult. In this work, the values are chosen in a way to cover the maximum load specification variation. The control compensator parameters can be set dynamically by the central controller for a better performance.
- The data bus used in the robot is based on a Serial Peripheral Interface (SPI), that can be switched to a standard Control Area Network (CAN 2.0A). The following change can provide more reliability of the data network and more robustness in the robot. Also the data network protocol can be switched to the standard CANopen communication protocol.
- Using 3D accelerometers and axis gyro rate sensors in the joint controllers, that are spread in the robot can be useful for more advanced control algorithms for the robot.
- The frontal ankle joint is based on a DC motor with maximum torque of 1Nm (using gear ratio). Using a stronger motor can improve the performance and the stability of the robot during frontal side swinging.

References

Andrews, G. R. (2000), Foundations of Multithreaded, Parallel, and Distributed Programming, Addison–Wesley, pp. 10.

Asada, H., I. Slotine, E. Jean-Jacques (1986), Robot Analysis and Control, Wiley.

Asama, H., H. Kurokawa, J. Ota, K. Sekiyama (2009), Distributed Autonomous Robotic Systems, Springer.

Asfour, T., F. Gyarfas, P. Azad, R. Dillmann (2006), an Integrated Humanoid Plattfrom for Sensory-Motor Control. In IEEE-RAS International Conference on Humanoid Robots (Humanoids 2006), Genoa, Italy.

Astroem, K., T. Haegglund (1995), PID Controller: Theory, Design, Tuning (2nd Edition), Instrument Society of America, North Carolina.

Austrian Micro System (2009), 360 Steps Programmable High Speed Magnetic Encoder AS5134.

Baltes, J., A. Byagowi, P. Kopacek, J. Anderson (2009), Teen Sized Humanoid Robot: Archie, Progress in Robotics, SpringerLink, FIRA RoboWorld Congress, Incheon, Korea.

Bill, G. (2002), a primer on distributed computing.

Bose, B.K. (2006), Power Electronics and Motor Drives: Advances and Trends, Elsevier Science Ltd.

Boulic, R., R. Mas, D. Thalmann (1996), A Robust Approach for the control of the center of mass with inverse kinematics, Elsevier Science Ltd.

Bradski, G., A. Kaehler (2008), Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media Inc.

Brown, S. (2004), Fundamentals of Digital Logic with VHDL Design (2nd edition), McGraw Hill.

Byagowi, A., P. Kopacek (2009), Using a humanoid robot for demining land mines, Supplementary Ways for Improving International Stability (SWIIS 2009), Politehnica University of Bucharest, Bucharest, Romania.

Chiasson, J. (2005), Modeling and High-Performance Control of Electric Machines, Wiley-IEEE press.

Chu, P. P. (2008), FPGA Prototyping by VHDL Examples, Wiley-Interscience.

Craig, J. (2005), Introduction to Robotics: Mechanics and Control (3rd Edition), Pearson Prentice Hall.

Denavit, J., R.S. Hartenberg (1955), a kinematic notation for lower-pair mechanisms based on matrices, Trans ASME J. Appl. Mech.

ELMO Motion Control (2009)., ELMO Web-based Motion Control Training course.

Emadi, A., A. Khaligh, Z. Nie, Y. J. Lee (2009), Integrated Power Electronic Converters and Digital Control, Taylor and Francis Group LLC, CRC Press.

Fadali, M. S. (2009), Digital Control Engineering: Analysis and Design, Elsevier Science Ltd.

Goldstein, H. (1980), Classical Mechanics (2nd edition), Addison-Wesley.

Gonzalez, R. C., R. E. Woods (2002), Digital Image Processing (2nd edition), Pearson Prentice Hall.

Hallinan, C. (2006), Embedded Linux Primer, Pearson Prentice Hall.

Honda (2002), Introducing a New ASIMO Featuring Intelligence Technology, Honda Motor Co., Ltd.

Ibrahim, D. (2006), Microcontroller Based Applied Digital Control, Department of Computer Engineering Near East University, John Wiley and Sons Ltd, Cyprus.

James, T. I., C. H. Lai (2006), Design and Control of a Humanoid Robot, IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China.

Katic, D., M. Vukobratovic (2004), Survey of Intelligent Control Techniques for Humanoid Robots, Springer Netherlands, pp. 117-141.

Kay, J. (2005), Introduction to Homogeneous Transformations and Robot Kinematic, Computer Science Department, Rowan University, New Jersey, USA.

Kim, S. K., S. Hong, D. Kim (2009), A Walking Motion Imitation Framework of a Humanoid Robot by Human Walking Recognition from IMU Motion Data, 9th IEEE-RAS International Conference on Humanoid Robots, Paris, France.

Kopacek, P. (2002), Final Report of Research Project, MAS for manufacturing automation with consideration of their application in small and medium-sized enterprises.

Landau, L., D. Lifschitz (1984), Electrodynamics of Continuous Media (2nd edition), Pergamon Press, Oxford, England.

Marion, J., S. Thornton (1995), Classical Dynamics of Systems and Particles (4th edition), Thomson.

Marshall, I. J. (2002), Active Balance Control for a Humanoid Robot, Department of Information Technology and Electrical Engineering, University of Queensland, Queensland, Australia.

Moreton, P. (1999), Industrial Brushless Servomotors, Newnes.

Moudgalya K. (2007), Digital Control, John Wiley and Sons Ltd.

Pickel, A. (2003), Control for a Biped Robot with Minimal Number of Actuator, Department of Electrical, Engineering, University of Applied Science Koblenz, Germany.

Rotenberg, S. (2005), Homogeneous Transformations, Computer Graphics, UCSD.

Rolf, C., F. Nikolaj, H. Rico (2007), Modeling and Control of a Biped Robot, Department of Control Engineering, Aalborg University, Aalborg, Denmark.

Sandhu, H. S. (2008), Running Small Motors with PIC Microcontrollers, McGraw Hill.

Shlomi, D. (2000), Self-Stabilization, MIT Press.

Spong, M., M. Vidyasagar (1989), Robot Dynamics and Control, John Wiley and Sons.

Sukumar, G. (2007), Distributed Systems – An Algorithmic Approach, Chapman & Hall/CRC.

Tabaczynski, M. (2006), Jacobian Solution to the Inverse Kinematics Problem, Tufts University, Medford, USA.

Takenaka, T. (2006), the control system for the Honda humanoid robot, Oxford University Press, Oxford, England.

Tang, Z., C. Zhou, Z. Sun (2003), Trajectory Planning for Smooth Transition of a Biped Robot, Proceedings of IEEE, International Conference on Robotics and Automation. Taipei, Taiwan.

Tolani, D., A. Goswami, N. Badler (2000), Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs, Computer and Information Science Department, University of Pennsylvania, Pennsylvania, USA.

Vukosavic, S. N. (2007), Digital Control of Electrical Drives, University of Belgrade, Springer.

Welman, C. (1993), Inverse Kinematics and Geometric constraints for articulated figure manipulation, Simon Fraser University, Vancouver, Canada.

Xie, M. (2003), Fundamentals of Robotics: Linking Perception to Action, World Scientific Publishing Company.

Xue, D., Y. Chen, D. P. Atherton (2007), Linear Feedback Control: Analysis and Design with MATLAB, Society of Industrial and Applied Mathematics, Philadelphia, USA.

Yaghmour, K., J. Masters, G. B. Yossef, P. Gerum (2008), Building Embedded Linux System (2nd edition), O'Reilly Media Inc.

Younkin, G. W. (2002), Industrial Servo Control Systems: fundamentals and Applications (2nd edition), Marcel Dekker.

Zelniker, E. (2001), Joint Control for an Autonomous Humanoid Robot, Department of information Technology and Electrical Engineering, University of Queensland, Queensland, Australia.

Appendix A

Matlab code for Trajectory planner, direct kinematics and Inverse Kinematics calculator for one leg:

---------- Trajectory planner (Elliptical trajectory) ----------

```
Lp=20;
Hp=16;

for t=1:100
    x(t)=Lp*cos(pi+((t-1)*pi/99));
    y(t)=Hp*sin((t-1)*pi/99)-52.45;
end

plot(x,y);
xlabel('Y-Axis (cm)');
ylabel('Z-Axis (cm)');
title('Elliptical Ankle desired trajectory');
```

---------- Trajectory planner (Trapezoidal trajectory) ----------

```
Lp=20;
Hp=16;
alpha=pi/6;
for t=1:100
    x(t)=t-1;
    if x(t) < Hp*cos(alpha)
       y(t)=x(t)*acot(alpha)-52.45;
    else
        if x(t) > 99-Hp*cos(alpha)
            y(t)=((99-x(t))*acot(alpha))-52.45;
        else
            y(t)=Hp-52.45;
        end
    end
end
x=(x*Lp*2/100)-Lp;
plot(x,y);
%legend('Desired Ankle trajectory');
xlabel('Y-Axis (cm)');
ylabel('Z-Axis (cm)');
title('Trapezoidal Ankle trajectory');
```

---------- Direct kinematics function ----------

```
function [xe,ye] = endeffector(stheta1,stheta2)
TibiaLength = 26;
ThighLength = 31;

xe=(ThighLength*sin(stheta1))-(TibiaLength*sin(stheta2-stheta1));
ye=-1*((ThighLength*cos(stheta1))+(TibiaLength*cos(stheta2-stheta1)));
```

---------- Inverse kinematics calculator ----------

```
TibiaLength = 26;
ThighLength = 31;
theta1=[1:100];
theta2=[1:100];

for i=1:100
    theta1(i)=0;
    theta2(i)=0;
end

theta1(1)=-10*(pi/180);
theta2(1)=20*(pi/180);

for i=2:100
            theta1(i)=theta1(i-1);
            theta2(i)=theta2(i-1);
            J_a=ThighLength*cos(theta1(i))+TibiaLength*cos(theta1(i)-
theta2(i));
            J_b=-TibiaLength*cos(theta2(i)-theta1(i));
            J_c=-ThighLength*sin(theta1(i))-TibiaLength*sin(theta1(i)-
theta2(i));
            J_d=-TibiaLength*sin(theta2(i)-theta1(i));
            J=[J_a J_b;J_c J_d];
            J_inv=inv(J);
            delta_theta=[0 0]';
            delta_pos=[0 0]';
            [xef(i),yef(i)]=endeffector(theta1(i),theta2(i));
            delta_pos(1) = x(i)-xef(i);
            delta_pos(2) = yef(i)-y(i);
            delta_theta=J_inv*delta_pos;
            theta1(i)=theta1(i)+delta_theta(1);
            theta2(i)=theta2(i)+delta_theta(2);
end

for i= 1:100
    [xf(i),yf(i)]=endeffector(theta1(i),theta2(i));
    errx(i)=xf(i)-x(i);
    erry(i)=yf(i)-y(i);
    err(i)=sqrt((errx(i)^2)+(erry(i)^2));
end

ttt=0:.05:4.95;
plot(ttt,err);
ylabel('Trajectory error (cm)');
xlabel('Time (sec)');
title('Swinging Ankle Trajectory error');
%break

plot(ttt,theta1);
ylabel('Hip Lateral joint (rad)');
xlabel('Time (sec)');
title('Hip Lateral joint time trajectory');
%break
```

```
plot(ttt,theta2);
ylabel('Swinging knee joint (rad)');
xlabel('Time (sec)');
title('Swinging joint time trajectory');
%break

plot(x,y,xf,yf,'r');
xlabel('Y-Axis (cm)');
ylabel('Z-Axis (cm)');
title('Desired and traversed trajectory');
legend('Desired Trajectory', 'Traversed Trajectory');
```

Image processing code used to track the patches (for finding the Traversed Trajectory) in Visual Studio 2008 and OpenCV 2.0 library:

```
#ifdef _CH_
#pragma package <opencv>
#endif

#ifndef _EiC
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include <ctype.h>
#include <stdio.h>
#include <math.h>
#endif

IplImage *gray=0,*img=0,*tra=0,*imgout=0;
CvMemStorage* storage;
IplImage *r_bin=0,*g_bin=0,*b_bin=0,*r_tr=0,*g_tr=0,*b_tr=0;
CvSeq* circles;
CvCapture *capture = 0;
CvVideoWriter *cvVideoWriter;
int temp[2][7];
int tempo[2][7];
int last_temp[2][7];
int dist,new_dist;
double fps;

int i,fl=1,ini=1,inw=1;

int main(int argc, char** argv)
{
```

```
   capture = cvCaptureFromAVI("video.avi");

   if( !capture )
    {
        fprintf(stderr,"Could not initialize capturing...\n");
        return -1;
    }

   img = cvQueryFrame( capture );

   fps = cvGetCaptureProperty(capture, CV_CAP_PROP_FPS);

   cvVideoWriter =
cvCreateVideoWriter("out.avi",CV_FOURCC('I','Y','U','V')
,fps,cvSize(img->width,img->height),1);

   tra = cvCreateImage(cvSize(img->width,img->height), 8, 3);

   gray = cvCreateImage(cvSize(img->width,img->height), IPL_DEPTH_8U,
1);

   storage = cvCreateMemStorage(0);

   r_bin = cvCreateImage( cvSize(img->width,img->height),IPL_DEPTH_8U,
1 );
   g_bin = cvCreateImage( cvSize(img->width,img->height),IPL_DEPTH_8U,
1 );
   b_bin = cvCreateImage( cvSize(img->width,img->height),IPL_DEPTH_8U,
1 );
   r_tr = cvCreateImage( cvSize(img->width,img->height),IPL_DEPTH_8U,
1 );
   g_tr = cvCreateImage( cvSize(img->width,img->height),IPL_DEPTH_8U,
1 );
   b_tr = cvCreateImage( cvSize(img->width,img->height),IPL_DEPTH_8U,
1 );
   imgout = cvCreateImage( cvSize(img->width,img-
>height),IPL_DEPTH_8U, img->nChannels );

   cvSplit(tra,r_tr,g_tr,b_tr,0);

   cvThreshold(r_tr,r_tr,254,255,CV_THRESH_BINARY);
   cvThreshold(g_tr,g_tr,254,255,CV_THRESH_BINARY);
   cvThreshold(b_tr,b_tr,254,255,CV_THRESH_BINARY);

   cvMerge(r_tr,g_tr,b_tr,0,tra);
```

```
for(;;)
{
   img = cvQueryFrame( capture );

   if(!img) break;

   cvCvtColor(img, imgout, CV_BGR2HSV);
   cvCvtColor(imgout, gray, CV_BGR2GRAY);

   cvThreshold(gray,gray,180,255,CV_THRESH_BINARY);

   cvSmooth(gray, gray, CV_GAUSSIAN, 9, 9, 0, 0);

   circles = cvHoughCircles(gray, storage, CV_HOUGH_GRADIENT, 1, 20,
200, 1, 10, 20);

    for (i = 0; i < circles->total; i++)
    {
         float* p = (float*)cvGetSeqElem( circles, i );
         cvCircle( img, cvPoint(cvRound(p[0]),cvRound(p[1])),
             3, CV_RGB(0,255,0), -1, 8, 0 );
       cvCircle( tra, cvPoint(cvRound(p[0]),cvRound(p[1])),
             1, CV_RGB(0,255,0), -1, 8, 0 );

         cvCircle( img, cvPoint(cvRound(p[0]),cvRound(p[1])),
             cvRound(p[2]), CV_RGB(255,255,0), 3, 8, 0 );
       printf("%d,%d\n",cvRound(p[0]),cvRound(p[1]));

    }

      cvSplit(img,r_bin,g_bin,b_bin,0);
      cvSplit(tra,r_tr,g_tr,b_tr,0);

      cvOr(r_bin,r_tr,r_bin,0);
      cvOr(g_bin,g_tr,g_bin,0);
      cvOr(b_bin,b_tr,b_bin,0);

      cvMerge(r_bin,g_bin,b_bin,0,img);

      cvNamedWindow( "Image", 1 );
      cvShowImage( "Image", img );


      cvWriteFrame(cvVideoWriter, img);
```

```
      if( cvWaitKey( 10 ) >= 0 )
                break;
   }

   cvReleaseCapture( &capture );

   cvReleaseVideoWriter(&cvVideoWriter);

   return 0;
}
```

PowerPC startup code used to upload the Linux kernel to the RAM memory written in EDK 10.1.

```
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xgpio.h"
#include "xuartns550_l.h"
#include "xuartns550.h"
#include "mmc-bitbang_ll.h"
#include "mmc_load_image.h"


#include "xexception_l.h"
#include "xintc.h"
#include "xtmrctr.h"


#include "mii-bitbang.h"
#include "marvell_88e1111.h"

#define PHY_ADDR   1

#define PHY_IDO_REG  0
#define PHY_ID1_REG  1
#define PHY_ID2_REG  2
#define PHY_ID3_REG  3
#define PHY_ID4_REG  4
```

```
#define PHY_ID16_REG  16

#define UART_BASEADDR XPAR_UARTNS550_0_BASEADDR
#define UART_CLOCK    XPAR_CPU_PPC405_CORE_CLOCK_FREQ_HZ
#define UART_BAUDRATE 19200                  /* real hardware */
#define uOLED_DEVICE_ID  XPAR_UOLED_BASEADDR


XGpio mii_gpio;
XUartNs550 uoled;       /* The instance of the UART Driver */
typedef void (*void_fn)(void *);
void_fn kernel_start;
char cmdline[256];

#define TMRCTR_DEVICE_ID  XPAR_TMRCTR_0_DEVICE_ID
#define TIMER_COUNTER_0     0

XTmrCtr TimerCounter; /* The instance of the timer counter */

void progress_bar(char left){
  XUartNs550_SendByte(uOLED_DEVICE_ID, 0x4c);
  XUartNs550_SendByte(uOLED_DEVICE_ID, 15+left);
  XUartNs550_SendByte(uOLED_DEVICE_ID, 75);
  XUartNs550_SendByte(uOLED_DEVICE_ID, 15+left);
  XUartNs550_SendByte(uOLED_DEVICE_ID, 105);
  XUartNs550_SendByte(uOLED_DEVICE_ID, 0);
  XUartNs550_SendByte(uOLED_DEVICE_ID, 0x1F);
//   while(0x06==XUartLite_RecvByte(uOLED_DEVICE_ID));
//   for (Delay = 0; Delay < 100000; Delay++);
}

void percent_bar(char val)
{

  uoled_send(0x73);
  if(val==100)
    uoled_send(8);
  else
```

```
    uoled_send(9);
  uoled_send(7);
  uoled_send(2);
  uoled_send(0xBB);
  uoled_send(0xBB);
  if(val==100)
    uoled_send(49);
  if(val>9)
    uoled_send(48+((val/10)%10));
  uoled_send(48+(val%10));
  uoled_send('%');
  uoled_send(0x00);
}


#define XPAR_SDRAM_8MX32_1_BASEADDR 0x00400000
unsigned int *mem_start = (unsigned int *) 0x00400000;


void dump() {
  Xuint8 temp;
  Xuint32 temp32;
  int i,j,k;
  k = 0;
  for (i=0;i<16;i++) {
    for (j=0;j<8;j++) {
      temp32 = XIo_In32(XPAR_SDRAM_8MX32_1_BASEADDR+k+0x0000);
      k+=4;
    }
  }
}


int docrc(int n) {
  Xuint32 temp;
  unsigned int len;
  unsigned int *dst;
  unsigned int *src;
```

```
  XTmrCtr *TmrCtrInstancePtr ;

  src = (unsigned int *)XPAR_SDRAM_8MX32_1_BASEADDR;
  temp = 0;
  len = n<<7;
  while(len--)
  {
    temp ^= *src++;
  }
  return temp;
}



void jump() {
  cmdline[0]='\0';
  kernel_start = (void_fn)mem_start;
  kernel_start(cmdline);
}



//void uoled_wait
//
void uoled_send(char uoled_dat){
  XUartNs550_SendByte(uOLED_DEVICE_ID, uoled_dat);
//   while(0x06==XUartLite_RecvByte(uOLED_DEVICE_ID));
//   for (Delay = 0; Delay < 100000; Delay++);
}

int main (void) {
  XStatus status;
  static XIntc intc;
  Xuint32 temp;
  Xuint32 temp2;
  u16   val, phy_id;
  XTmrCtr *TmrCtrInstancePtr = &TimerCounter;
  int i,j,k;
```

```
  unsigned int arg32;
  unsigned int RCA;
  int p;
  volatile int Delay,d;

  XUartNs550_SetBaud(XPAR_RS232_BASEADDR, XPAR_UARTNS550_0_CLOCK_FREQ_HZ,
19200);
  XUartNs550_mSetLineControlReg(XPAR_RS232_BASEADDR, XUN_LCR_8_DATA_BITS);

  MiiGpio_Init(&mii_gpio);
  marvell_phy_setvectors((PhyRead_t*)MiiGpio_PhyRead,
                  (PhyWrite_t*)MiiGpio_PhyWrite);

  XUartNs550_SetBaud(XPAR_IMU_UNIT_BASEADDR, XPAR_UARTNS550_0_CLOCK_FREQ_HZ,
115200);
  XUartNs550_mSetLineControlReg(XPAR_IMU_UNIT_BASEADDR, XUN_LCR_8_DATA_BITS);

  XUartNs550_SetBaud(uOLED_DEVICE_ID, XPAR_UARTNS550_0_CLOCK_FREQ_HZ, 115200);
  XUartNs550_mSetLineControlReg(uOLED_DEVICE_ID, XUN_LCR_8_DATA_BITS);
  status = XUartNs550_Initialize(&uoled, uOLED_DEVICE_ID);
  if (status != XST_SUCCESS)
  {
     xil_printf("\n\r uOLED Device Failed...\n\r");
  }

  XIo_Out32(uOLED_DEVICE_ID + 0x1010,XUN_OPTION_ASSERT_RTS);

  for(d=0;d<2;d++)
  {
    *((volatile unsigned int *) XPAR_LEDS_BASEADDR) = 0x00000001<<d;
    for (Delay = 0; Delay < 300000; Delay++);
  }

  XUartNs550_SendByte(uOLED_DEVICE_ID, 0x55);
  for (Delay = 0; Delay < 300000; Delay++);

  XUartNs550_SendByte(uOLED_DEVICE_ID, 0x45);
  for (Delay = 0; Delay < 300000; Delay++);
```

```
uoled_send(0x53);
uoled_send(0x00);
uoled_send(0x00);
uoled_send(0x02);
uoled_send(0xBB);
uoled_send(0xBB);
uoled_send(0x01);
uoled_send(0x01);
uoled_send('S');
uoled_send('t');
uoled_send('a');
uoled_send('r');
uoled_send('t');
uoled_send('i');
uoled_send('n');
uoled_send('g');
uoled_send('.');
uoled_send('.');
uoled_send('.');
uoled_send(0x00);
for (Delay = 0; Delay < 200000; Delay++);

for(d=2;d<4;d++)
{
  *((volatile unsigned int *) XPAR_LEDS_BASEADDR) = 0x00000001<<d;
  for (Delay = 0; Delay < 300000; Delay++);
}

xil_printf("\n\r ARCHIE O/S Loader v1.5\n\r");

uoled_send(0x53);
uoled_send(0x13);
uoled_send(0x20);
uoled_send(0x02);
uoled_send(0xAB);
uoled_send(0xAB);
uoled_send(0x01);
```

```
uoled_send(0x02);
uoled_send('A');
uoled_send('r');
uoled_send('c');
uoled_send('h');
uoled_send('i');
uoled_send('e');
uoled_send('L');
uoled_send('i');
uoled_send('n');
uoled_send('u');
uoled_send('x');
uoled_send(' ');
uoled_send('2');
uoled_send('.');
uoled_send('6');
uoled_send(0x00);
for (Delay = 0; Delay < 200000; Delay++);

uoled_send(0x72);
uoled_send(13);
uoled_send(73);
uoled_send(145);
uoled_send(107);
uoled_send(0xC8);
uoled_send(0x00);

for (Delay = 0; Delay < 2000; Delay++);

uoled_send(0x4f);
uoled_send(1);

val = marvell_phy_detected(&mii_gpio, PHY_ADDR);
if (val == XFALSE) {
   print("Mii Device not detected.\n\r");
}
```

```
  XIo_Out32(XPAR_LEDS_BASEADDR, 0x00000001);    //init value for LED

  *((volatile unsigned int *) XPAR_LEDS_BASEADDR) = 0x00;

  while (1) {
    if (CardSense()) {
      print("\n\rCard inserted");
      mmc_init();

      load_boot_image(0, XPAR_SDRAM_8MX32_1_BASEADDR);

      dump();

      mmc_send_command(12, 0);
      Init_80_Clocks();
      print("\n\rStarting image from Memory\n\r");
      jump();
      print("PANIC: Kernel returned!");
    } else {
      print("Card missing...\n\r");
      while (!CardSense()) {}
    }
  }

  return 0;
}
```

The Embedded system developed for the central controller main processing system in EDK.

```
PARAMETER VERSION = 2.1.0


 PORT RXD_0 = fpga_0_RS232_sin, DIR = I
 PORT TXD_0 = fpga_0_RS232_sout, DIR = O
 PORT SD_IO = fpga_0_Generic_GPIO_GPIO_IO, DIR = IO, VEC = [0:3]
 PORT LEDs_pin = fpga_0_LEDS_GPIO_d_out, DIR = O, VEC = [0:3]
 PORT PHY_tx_clk = fpga_0_Generic_Ethernet_10_100_PHY_tx_clk, DIR = I
 PORT PHY_rx_clk = fpga_0_Generic_Ethernet_10_100_PHY_rx_clk, DIR = I
 PORT PHY_crs = fpga_0_Generic_Ethernet_10_100_PHY_crs, DIR = I
 PORT PHY_dv = fpga_0_Generic_Ethernet_10_100_PHY_dv, DIR = I
```

```
 PORT PHY_rx_data = fpga_0_Generic_Ethernet_10_100_PHY_rx_data, DIR =
I, VEC = [3:0]
 PORT PHY_col = fpga_0_Generic_Ethernet_10_100_PHY_col, DIR = I
 PORT PHY_rx_er = fpga_0_Generic_Ethernet_10_100_PHY_rx_er, DIR = I
 PORT PHY_tx_en = fpga_0_Generic_Ethernet_10_100_PHY_tx_en, DIR = O
 PORT PHY_tx_data = fpga_0_Generic_Ethernet_10_100_PHY_tx_data, DIR =
O, VEC = [3:0]
 PORT PHY_rst_n = fpga_0_Generic_Ethernet_10_100_PHY_rst_n, DIR = O
 PORT I2C_SCL = fpga_0_Generic_IIC_Bus_Scl, DIR = IO
 PORT I2C_SDA = fpga_0_Generic_IIC_Bus_Sda, DIR = IO
 PORT SPI_FLASH_SCK = fpga_0_Generic_SPI_SCK, DIR = IO
 PORT SPI_FLASH_MOSI = fpga_0_Generic_SPI_MOSI, DIR = IO
 PORT SPI_FLASH_MISO = fpga_0_Generic_SPI_MISO, DIR = IO
 PORT SDRAM_A = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_Addr, DIR = O, VEC =
[11:0]
 PORT SDRAM_BA = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_BankAddr, DIR = O, VEC
= [1:0]
 PORT SDRAM_CASn = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_CAS_n, DIR = O
 PORT SDRAM_RASn = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_RAS_n, DIR = O
 PORT SDRAM_WEn = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_WE_n, DIR = O
 PORT SDRAM_CKE = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_CE, DIR = O
 PORT SDRAM_CSn = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_CS_n, DIR = O
 PORT SDRAM_Clk = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_Clk, DIR = O
 PORT SDRAM_DQM = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_DM, DIR = O, VEC =
[3:0]
 PORT SDRAM_DQ = SDRAM_DQ, DIR = IO, VEC = [31:0]
 PORT sys_clk = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 12000000
 PORT sys_rst = sys_rst_s, DIR = I, RST_POLARITY = 0, SIGIS = RST
 PORT SDRAM_A12 = net_vcc, DIR = O
 PORT PBs = fpga_0_Push_Buttons_GPIO_in, DIR = I, VEC = [0:4]
 PORT MII_MDC_MDIO_PIN = MII_MDC_MDIO_GPIO_GPIO_IO, DIR = IO, VEC =
[0:1]
 PORT PHY_tx_er = net_gnd, DIR = O
 PORT TXOLED = uOLED_TX, DIR = O
 PORT RXOLED = uOLED_RX, DIR = I
 PORT MOSI1 = SPI_1_MOSI, DIR = IO
 PORT MISO1 = SPI_1_MISO, DIR = IO
 PORT SCLK1 = SPI_1_SCK, DIR = IO
 PORT MOSI2 = SPI_2_MOSI, DIR = IO
 PORT MISO2 = SPI_2_MISO, DIR = IO
 PORT SCLK2 = SPI_2_SCK, DIR = IO
 PORT MOSI3 = SPI_3_MOSI, DIR = IO
 PORT MISO3 = SPI_3_MISO, DIR = IO
 PORT SCLK3 = SPI_3_SCK, DIR = IO
 PORT IMU_sout = IMU_unit_sout, DIR = O
 PORT IMU_sin = IMU_unit_sin, DIR = I
 PORT uOLED_rts = not_gate_Res, DIR = O, VEC = [0:0]


BEGIN ppc405_virtex4
 PARAMETER INSTANCE = ppc405_0
 PARAMETER HW_VER = 2.01.a
```

```
 PARAMETER C_FASTEST_PLB_CLOCK = DPLB1
 PARAMETER C_IDCR_BASEADDR = 0b0100000000
 PARAMETER C_IDCR_HIGHADDR = 0b0111111111
 BUS_INTERFACE JTAGPPC = jtagppc_0_0
 BUS_INTERFACE IPLB0 = plb
 BUS_INTERFACE DPLB0 = plb
 BUS_INTERFACE IPLB1 = ppc405_0_iplb1
 BUS_INTERFACE DPLB1 = ppc405_0_dplb1
 BUS_INTERFACE RESETPPC = ppc_reset_bus
 PORT CPMC405CLOCK = sys_clk_s
 PORT EICC405EXTINPUTIRQ = EICC405EXTINPUTIRQ
END

BEGIN jtagppc_cntlr
 PARAMETER INSTANCE = jtagppc_0
 PARAMETER HW_VER = 2.01.c
 BUS_INTERFACE JTAGPPC0 = jtagppc_0_0
END

BEGIN plb_v46
 PARAMETER INSTANCE = plb
 PARAMETER C_DCR_INTFCE = 0
 PARAMETER C_NUM_CLK_PLB2OPB_REARB = 100
 PARAMETER HW_VER = 1.03.a
 PORT PLB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END

BEGIN xps_bram_if_cntlr
 PARAMETER INSTANCE = xps_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SPLB_NATIVE_DWIDTH = 64
 PARAMETER C_BASEADDR = 0xffff8000
 PARAMETER C_HIGHADDR = 0xffffffff
 BUS_INTERFACE SPLB = plb
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_1_port
END

BEGIN bram_block
 PARAMETER INSTANCE = plb_bram_if_cntlr_1_bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_1_port
END

BEGIN xps_ethernetlite
 PARAMETER INSTANCE = Generic_Ethernet_10_100
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_SPLB_CLK_PERIOD_PS = 13888
 PARAMETER C_BASEADDR = 0x81000000
 PARAMETER C_HIGHADDR = 0x8100ffff
 BUS_INTERFACE SPLB = plb
 PORT PHY_tx_clk = fpga_0_Generic_Ethernet_10_100_PHY_tx_clk
```

```
 PORT PHY_rx_clk = fpga_0_Generic_Ethernet_10_100_PHY_rx_clk
 PORT PHY_crs = fpga_0_Generic_Ethernet_10_100_PHY_crs
 PORT PHY_dv = fpga_0_Generic_Ethernet_10_100_PHY_dv
 PORT PHY_rx_data = fpga_0_Generic_Ethernet_10_100_PHY_rx_data
 PORT PHY_col = fpga_0_Generic_Ethernet_10_100_PHY_col
 PORT PHY_rx_er = fpga_0_Generic_Ethernet_10_100_PHY_rx_er
 PORT PHY_tx_en = fpga_0_Generic_Ethernet_10_100_PHY_tx_en
 PORT PHY_tx_data = fpga_0_Generic_Ethernet_10_100_PHY_tx_data
 PORT PHY_rst_n = fpga_0_Generic_Ethernet_10_100_PHY_rst_n
 PORT IP2INTC_Irpt = Generic_Ethernet_10_100_IP2INTC_Irpt
END

BEGIN xps_gpio
 PARAMETER INSTANCE = SD
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_GPIO_WIDTH = 4
 PARAMETER C_IS_BIDIR = 1
 PARAMETER C_IS_DUAL = 0
 PARAMETER C_BASEADDR = 0x81400000
 PARAMETER C_HIGHADDR = 0x8140ffff
 BUS_INTERFACE SPLB = plb
 PORT GPIO_IO = fpga_0_Generic_GPIO_GPIO_IO
END

BEGIN xps_gpio
 PARAMETER INSTANCE = LEDS
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_GPIO_WIDTH = 4
 PARAMETER C_IS_DUAL = 0
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_IS_BIDIR = 0
 PARAMETER C_BASEADDR = 0x81460000
 PARAMETER C_HIGHADDR = 0x8146ffff
 BUS_INTERFACE SPLB = plb
 PORT GPIO_d_out = fpga_0_LEDS_GPIO_d_out
END

BEGIN xps_gpio
 PARAMETER INSTANCE = Push_Buttons
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_GPIO_WIDTH = 5
 PARAMETER C_IS_DUAL = 0
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_IS_BIDIR = 0
 PARAMETER C_BASEADDR = 0x81420000
 PARAMETER C_HIGHADDR = 0x8142ffff
 BUS_INTERFACE SPLB = plb
 PORT GPIO_in = fpga_0_Push_Buttons_GPIO_in
END

BEGIN xps_iic
```

```
 PARAMETER INSTANCE = Generic_IIC_Bus
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_CLK_FREQ = 72000000
 PARAMETER C_BASEADDR = 0x81600000
 PARAMETER C_HIGHADDR = 0x8160ffff
 BUS_INTERFACE SPLB = plb
 PORT Scl = fpga_0_Generic_IIC_Bus_Scl
 PORT Sda = fpga_0_Generic_IIC_Bus_Sda
 PORT IIC2INTC_Irpt = Generic_IIC_Bus_IIC2INTC_Irpt
END

BEGIN mpmc
 PARAMETER INSTANCE = SDR_SDRAM_CUSTOM
 PARAMETER HW_VER = 4.03.a
 PARAMETER C_NUM_PORTS = 2
 PARAMETER C_MEM_PARTNO = CUSTOM
 PARAMETER C_MEM_TYPE = SDRAM
 PARAMETER C_MEM_CE_WIDTH = 1
 PARAMETER C_MEM_CS_N_WIDTH = 1
 PARAMETER C_MEM_CLK_WIDTH = 1
 PARAMETER C_MEM_NUM_RANKS = 1
 PARAMETER C_MEM_DATA_WIDTH = 32
 PARAMETER C_PIM1_BASETYPE = 2
 PARAMETER C_MPMC_CLK0_PERIOD_PS = 13888
 PARAMETER C_MEM_PART_DATA_DEPTH = 128
 PARAMETER C_MEM_PART_DATA_WIDTH = 32
 PARAMETER C_MEM_PART_NUM_BANK_BITS = 2
 PARAMETER C_MEM_PART_NUM_ROW_BITS = 12
 PARAMETER C_MEM_PART_NUM_COL_BITS = 8
 PARAMETER C_MEM_PART_CAS_A_FMAX = 105
 PARAMETER C_MEM_PART_CAS_A = 3
 PARAMETER C_MEM_PART_TRAS = 60000
 PARAMETER C_MEM_PART_TRASMAX = 100000000
 PARAMETER C_MEM_PART_TRC = 84000
 PARAMETER C_MEM_PART_CAS_B_FMAX = 83
 PARAMETER C_MEM_PART_CAS_B = 2
 PARAMETER C_MEM_PART_TWR = 15000
 PARAMETER C_MEM_PART_CAS_C_FMAX = 35
 PARAMETER C_MEM_PART_CAS_C = 1
 PARAMETER C_MEM_PART_TRRD = 19000
 PARAMETER C_MEM_PART_TRCD = 24000
 PARAMETER C_MEM_PART_TREFI = 7800000
 PARAMETER C_MEM_PART_TRFC = 75000
 PARAMETER C_MEM_PART_TRP = 24000
 PARAMETER C_MPMC_BASEADDR = 0x00000000
 PARAMETER C_MPMC_HIGHADDR = 0x00ffffff
 BUS_INTERFACE SPLB0 = ppc405_0_iplb1
 BUS_INTERFACE SPLB1 = ppc405_0_dplb1
 PORT SDRAM_Addr = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_Addr
 PORT SDRAM_BankAddr = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_BankAddr
 PORT SDRAM_CAS_n = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_CAS_n
 PORT SDRAM_RAS_n = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_RAS_n
```

```
 PORT SDRAM_WE_n = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_WE_n
 PORT SDRAM_CE = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_CE
 PORT SDRAM_CS_n = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_CS_n
 PORT SDRAM_Clk = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_Clk
 PORT SDRAM_DM = fpga_0_SDR_SDRAM_CUSTOM_SDRAM_DM
 PORT SDRAM_DQ = SDRAM_DQ
 PORT MPMC_Clk0 = sys_clk_s
 PORT MPMC_Clk90 = SDR_SDRAM_CUSTOM_mpmc_clk_90_s
 PORT MPMC_Rst = sys_periph_reset
END

BEGIN xps_spi
 PARAMETER INSTANCE = Generic_SPI
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_FIFO_EXIST = 1
 PARAMETER C_NUM_SS_BITS = 1
 PARAMETER C_NUM_TRANSFER_BITS = 8
 PARAMETER C_SCK_RATIO = 128
 PARAMETER C_BASEADDR = 0x83c12000
 PARAMETER C_HIGHADDR = 0x83c1207f
 BUS_INTERFACE SPLB = plb
 PORT SCK = fpga_0_Generic_SPI_SCK
 PORT MOSI = fpga_0_Generic_SPI_MOSI
 PORT MISO = fpga_0_Generic_SPI_MISO
 PORT IP2INTC_Irpt = Generic_SPI_IP2INTC_Irpt
END

BEGIN xps_uart16550
 PARAMETER INSTANCE = RS232
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_IS_A_16550 = 1
 PARAMETER C_BASEADDR = 0x83e20000
 PARAMETER C_HIGHADDR = 0x83e2ffff
 BUS_INTERFACE SPLB = plb
 PORT sin = fpga_0_RS232_sin
 PORT sout = fpga_0_RS232_sout
 PORT ctsN = net_gnd
 PORT IP2INTC_Irpt = RS232_IP2INTC_Irpt
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_COUNT_WIDTH = 32
 PARAMETER C_ONE_TIMER_ONLY = 1
 PARAMETER C_BASEADDR = 0x83c00000
 PARAMETER C_HIGHADDR = 0x83c0ffff
 BUS_INTERFACE SPLB = plb
 PORT Interrupt = xps_timer_1_Interrupt
END

BEGIN plb_v46
```

```
 PARAMETER INSTANCE = ppc405_0_iplb1
 PARAMETER HW_VER = 1.03.a
 PORT PLB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END

BEGIN plb_v46
 PARAMETER INSTANCE = ppc405_0_dplb1
 PARAMETER HW_VER = 1.03.a
 PORT PLB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END

BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 BUS_INTERFACE RESETPPC0 = ppc_reset_bus
 PORT Slowest_sync_clk = sys_clk_s
 PORT Dcm_locked = net_vcc
 PORT Ext_Reset_In = sys_rst_s
 PORT Bus_Struct_Reset = sys_bus_reset
 PORT Peripheral_Reset = sys_periph_reset
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = plb
 PORT Irq = EICC405EXTINPUTIRQ
 PORT Intr =
xps_timer_1_Interrupt&Generic_Ethernet_10_100_IP2INTC_Irpt&RS232_IP2IN
TC_Irpt&SPI_1_IP2INTC_Irpt&SPI_2_IP2INTC_Irpt&SPI_3_IP2INTC_Irpt&IMU_u
nit_IP2INTC_Irpt&uOLED_IP2INTC_Irpt&Generic_IIC_Bus_IIC2INTC_Irpt&Gene
ric_SPI_IP2INTC_Irpt
END

BEGIN dcm_module
 PARAMETER INSTANCE = clock_72Mhz
 PARAMETER HW_VER = 1.00.d
 PARAMETER C_CLKOUT_PHASE_SHIFT = FIXED
 PARAMETER C_STARTUP_WAIT = TRUE
 PARAMETER C_CLKFX_MULTIPLY = 6
 PARAMETER C_CLKIN_PERIOD = 83.0
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT CLKFX = sys_clk_s
 PORT RST = sys_rst_s
 PORT CLKIN = dcm_clk_s
 PORT CLKFB = clock_72Mhz_90ph_CLK270
END
```

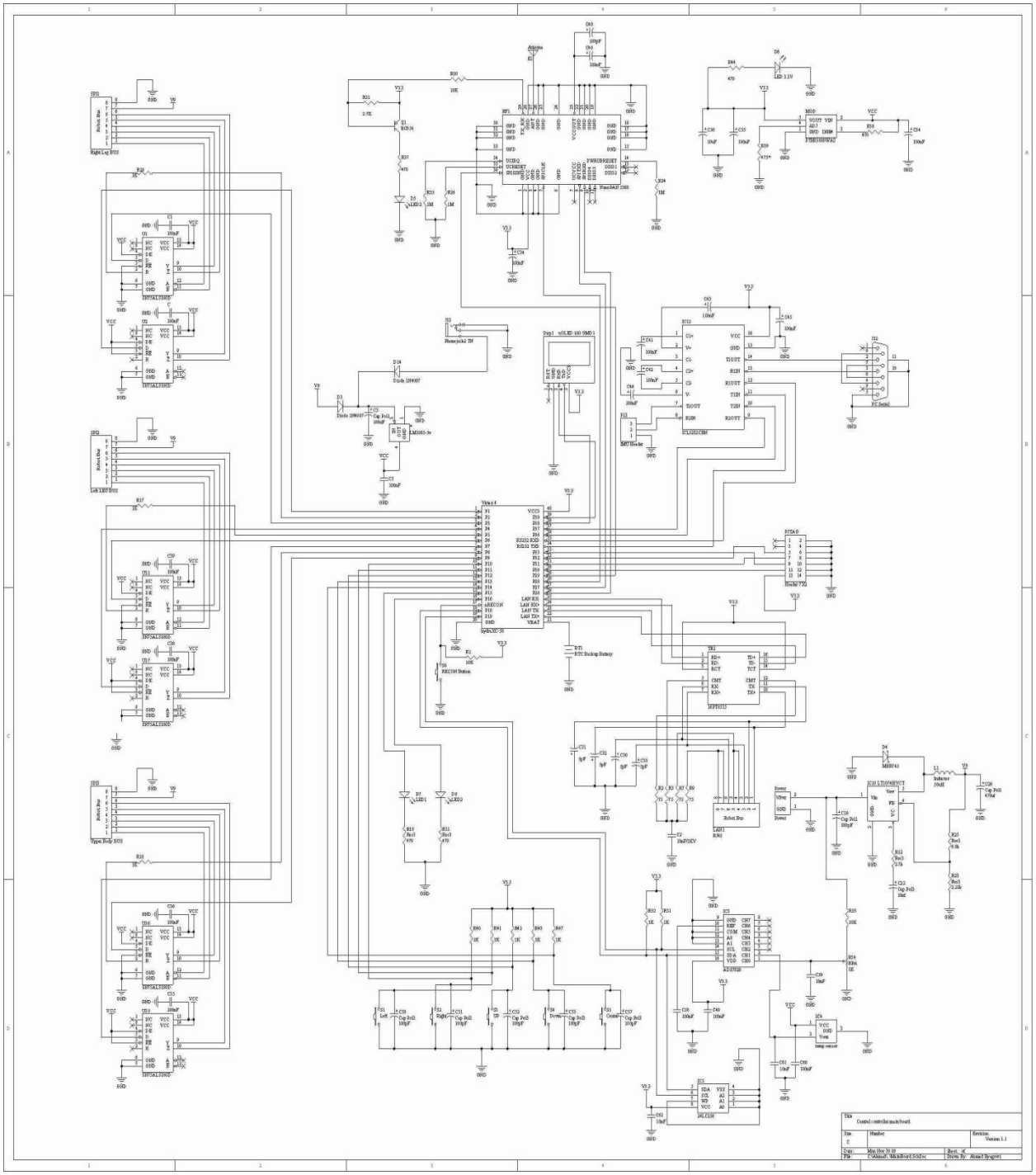
```
BEGIN dcm_module
 PARAMETER INSTANCE = clock_72Mhz_90ph
 PARAMETER HW_VER = 1.00.d
 PARAMETER C_CLKOUT_PHASE_SHIFT = FIXED
 PARAMETER C_STARTUP_WAIT = TRUE
 PARAMETER C_PHASE_SHIFT = 90
 PARAMETER C_CLKFX_MULTIPLY = 6
 PARAMETER C_CLKIN_PERIOD = 83.0
 PARAMETER C_EXT_RESET_HIGH = 0
 PORT RST = sys_rst_s
 PORT CLKIN = dcm_clk_s
 PORT CLK270 = clock_72Mhz_90ph_CLK270
 PORT CLKFX = SDR_SDRAM_CUSTOM_mpmc_clk_90_s
END

BEGIN xps_gpio
 PARAMETER INSTANCE = MII_MDC_MDIO_GPIO
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_GPIO_WIDTH = 2
 PARAMETER C_BASEADDR = 0x81440000
 PARAMETER C_HIGHADDR = 0x8144ffff
 BUS_INTERFACE SPLB = plb
 PORT GPIO_IO = MII_MDC_MDIO_GPIO_GPIO_IO
END

BEGIN xps_spi
 PARAMETER INSTANCE = SPI_1
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_NUM_TRANSFER_BITS = 16
 PARAMETER C_FIFO_EXIST = 0
 PARAMETER C_BASEADDR = 0x83c1c080
 PARAMETER C_HIGHADDR = 0x83c1c0ff
 BUS_INTERFACE SPLB = plb
 PORT MOSI = SPI_1_MOSI
 PORT MISO = SPI_1_MISO
 PORT SCK = SPI_1_SCK
 PORT IP2INTC_Irpt = SPI_1_IP2INTC_Irpt
END

BEGIN xps_spi
 PARAMETER INSTANCE = SPI_2
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_NUM_TRANSFER_BITS = 16
 PARAMETER C_FIFO_EXIST = 0
 PARAMETER C_BASEADDR = 0x83c14000
 PARAMETER C_HIGHADDR = 0x83c1407f
 BUS_INTERFACE SPLB = plb
 PORT MOSI = SPI_2_MOSI
 PORT MISO = SPI_2_MISO
 PORT SCK = SPI_2_SCK
 PORT IP2INTC_Irpt = SPI_2_IP2INTC_Irpt
```

```
END

BEGIN xps_spi
 PARAMETER INSTANCE = SPI_3
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_NUM_TRANSFER_BITS = 16
 PARAMETER C_FIFO_EXIST = 0
 PARAMETER C_BASEADDR = 0x83c18000
 PARAMETER C_HIGHADDR = 0x83c1807f
 BUS_INTERFACE SPLB = plb
 PORT MOSI = SPI_3_MOSI
 PORT MISO = SPI_3_MISO
 PORT SCK = SPI_3_SCK
 PORT IP2INTC_Irpt = SPI_3_IP2INTC_Irpt
END

BEGIN xps_uart16550
 PARAMETER INSTANCE = IMU_unit
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_BASEADDR = 0x83e40000
 PARAMETER C_HIGHADDR = 0x83e4ffff
 BUS_INTERFACE SPLB = plb
 PORT ctsN = net_gnd
 PORT IP2INTC_Irpt = IMU_unit_IP2INTC_Irpt
 PORT sout = IMU_unit_sout
 PORT sin = IMU_unit_sin
END

BEGIN xps_uart16550
 PARAMETER INSTANCE = uOLED
 PARAMETER HW_VER = 2.00.b
 PARAMETER C_BASEADDR = 0x83e00000
 PARAMETER C_HIGHADDR = 0x83e0ffff
 BUS_INTERFACE SPLB = plb
 PORT ctsN = net_gnd
 PORT sout = uOLED_TX
 PORT sin = uOLED_RX
 PORT IP2INTC_Irpt = uOLED_IP2INTC_Irpt
 PORT rtsN = uOLED_rtsN
END

BEGIN util_vector_logic
 PARAMETER INSTANCE = not_gate
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_OPERATION = not
 PARAMETER C_SIZE = 1
 PORT Res = not_gate_Res
 PORT Op1 = uOLED_rtsN
END
```

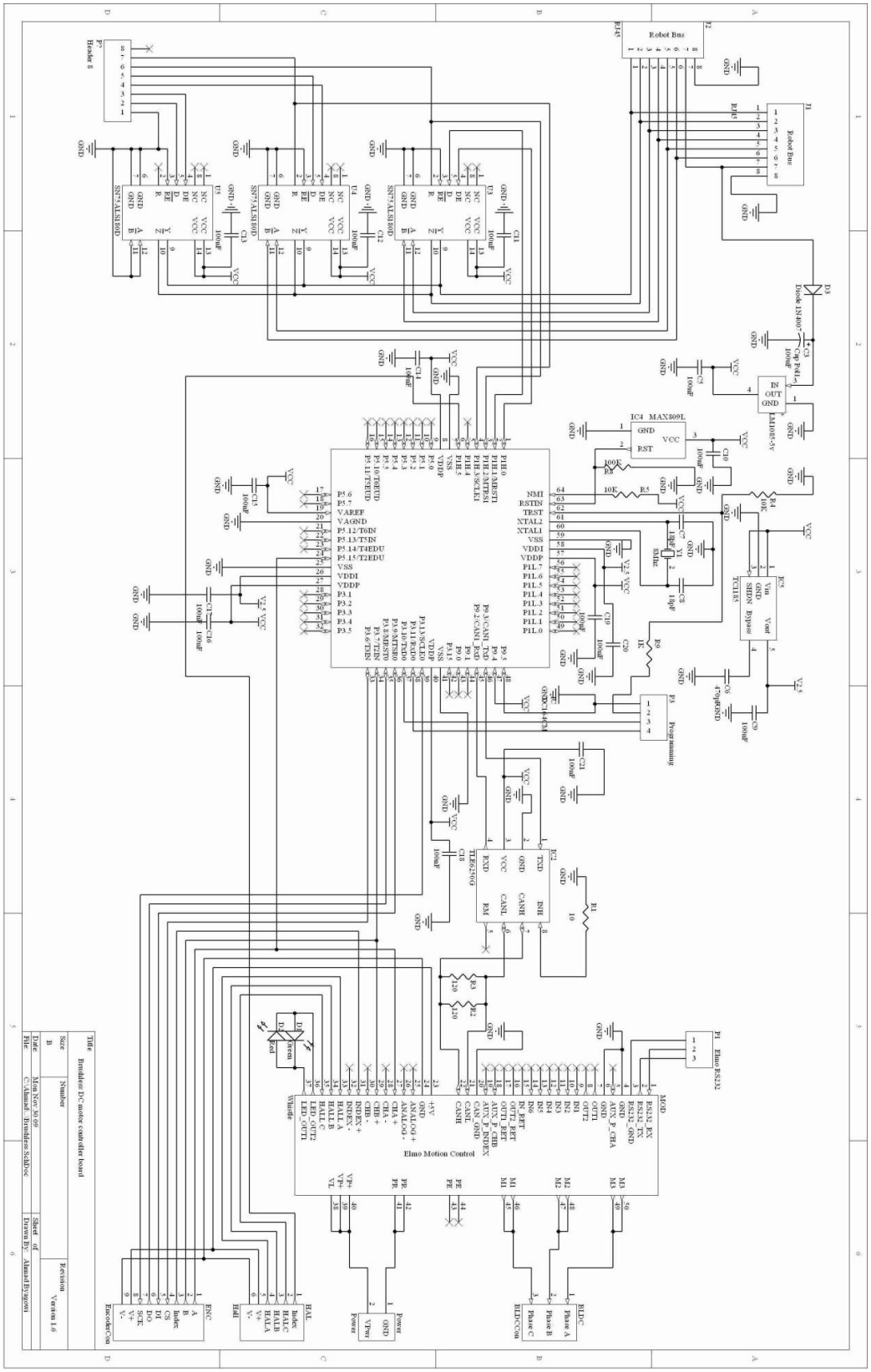
Appendix B

Schematic of the designed circuits in project

The main board's schematic



The Brushless motor controller's schematic:



The DC motor controller's schematic: