INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Humanoid Robot

Development of a simulation environment of an entertainment humanoid robot

Pedro Daniel Dinis Teodoro

Dissertação para obtenção do Grau de Mestre em

Engenharia Mecânica

Júri

Presidente:	Professor José Manuel Gutierrez Sá da Costa
Orientador:	Professor Miguel Afonso Dias de Ayala Botto
Co-orientador:	Professor Jorge Manuel Mateus Martins
Vogal:	Professor João Miguel da Costa Sousa

September - 2007

Este trabalho reflecte as ideias dos seus
autores que, eventualmente, poderão não
coincidir com as do Instituto Superior Técnico.

Abstract

This dissertation was developed in collaboration with Robosavvy Ltd and boosted the creation of the Humanoid Robotics Laboratory of IDMEC-Center of Intelligent Systems, at Instituto Superior Técnico (http://humanoids.dem.ist.utl.pt/). The developments presented include: i) the software development for interfacing the Matlab® Real Time Workshop Toolbox with the Bioloid humanoid robot servos; ii) the identification of the internal and external dynamic parameter of the humanoid servos and structure, respectively; iii) the dynamics modeling and simulation of the humanoid robot using the SimMechanics® and Virtual Reality Toolbox®; iv) the deduction of the equations of motion for an underactuated n-link inverted pendulum. The main objective of the Humanoid Robotics Laboratory, for the time being, is to develop a humanoid robot able to make complex motions like walking, running and jumping through real-time feedback control techniques. This dissertation presents a LQR controller for the simulation and control of the humanoid robot doing the handstand on a high bar, by considering it as an underactuated 3-link inverted pendulum.

Keywords: Humanoid Identification, Servo Identification, Humanoid Simulation, Linear Quadratic Regulator (LQR), Underactuated Inverted Pendulum, Non-Minimum Phase-System.

Resumo

Esta dissertação foi desenvolvida em colaboração com a Robosavvy Ltd e proporcionou a criação do laboratório de humanóides robóticos (Humanoid Robotics Laboratory) do IDMEC-Centro de Sistemas Inteligentes do Instituto Superior Técnico (http://humanoids.dem.ist.utl.pt/). Os progressos realizados incluem: i) o desenvolvimento de software para interface entre o Matlab® Real Time Workshop Toolbox com o robô humanoide Bioloid; ii) a identificação dos parâmetros dinâmicos internos e externos dos servos e da estrutura do humanóide, respectivamente; iii) a modelação dinâmica e a simulação do robô humanóide usando o SimMechanics® e a Virtual Reality Toolbox®; iv) a dedução das equações de movimento para um pêndulo invertido subactuado de n-barras. O objectivo do laboratório de humanóides robóticos, neste momento, é o desenvolvimento de um robô humanóide capaz de exercer movimentos complexos como andar, correr e saltar através de técnicas de controlo por feedback em tempo real. Esta dissertação apresenta um controlador LQR para a simulação e controlo de um robô humanóide a fazer o pino sobre uma barra, comportando-se como um pêndulo invertido subactuado de 3-barras.

Keywords: Identificação de Humanóides, Identificação de Servos, Simulação de Humanóides,Regulador Linear Quadrático (LQR), Pêndulo Invertido Subactuado, Sistema de Fase Não Mínima.

Acknowledgments

This thesis would never be done without the full support, orientation, dedication and availability of my supervisors, Prof. Miguel Ayala Botto and Prof. Jorge Martins. To them, my deepest gratitude.

I would like to express my gratitude for Professor Carlos Cardeira for his help in the serial communication between the PC and the humanoid.

I would like to thank Limor Schweitzer, for his inestimable dedication and help in every phase of the project.

I wish to thank all my colleagues from the DEM, for the friendly work environment they provided, and for the help they gave me in the many different occasions.

Least but not last, I shall express, from the bottom of my heart, my deepest gratitude to my parents and family for the support, dedication, comprehension and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> We are all, human and humanoid alike,
> whether made of flesh or of metal,
> basically just sociable machines.
>
> Robin Marantz Henig

A long-standing desire that human-like robots could coexist with human beings has made the researchers think that the humanoid robotics industry will be a leading industry in the twentieth-first century (Kim et al., 2007). This thought comes from the fact that technology is finally getting ready for this purpose. Fastest micro-processors, super computers, high-torque servo-actuators, precise sensors along with new advances in control techniques, artificial intelligent and artificial sound/vision recognition, all embedded in better and better mechanical design machines made the believe that this dream might became true in a nearly future. But, humanoid robots will not only be able to socialize with the human-being but will also be able to replace him even in the tedious and dangerous tasks, ranging from rescuing situations to interplanetary exploration (Ramamoorthy, 2007).

1.1 State of Art

ASIMO from Honda (Figure 1.1a) is until this moment the most advanced humanoid robot ever created, with voice, vision and gesture recognition (Sakagami et al., 2002), besides dynamic advanced locomotion control that make it able to walk, run, climb stairs and avoid static and dynamic obstacles while walking (Chestnutt et al., 2005; Takenaka, 2006). In fact, locomotion is the main characteristic studied in humanoid robotics for two reasons; first, a humanoid can only resemble like a human if it is able to move like him; second and as cited by Wolpert (Wolpert et al., 2001), *"Movement provides the only means we have to interact with both the world and other people."*.

And only after achieving the natural walking and locomotion of a humanoid in the human environment, are humanoid robots able to learn how to interact with it and socialize with humans, making use of all of its artificial intelligent.

Walking locomotion is not a trivial concept for the human understanding. In fact, only recently studies (Sockol et al., 2007) demonstrated, by comparing the oxygen consumption of humans and chimpanzees while walking on a treadmill, that the human-being evolved to walk upright in two legs (bipedalism) since it makes the walking far more efficient in terms of wasting energy. Understanding the natural and smooth walking of a human is also another challenge since generating a stable walking motion for this multi-body system, which is highly nonlinear, is a very complex one. Hence, studies were done around this subject and successfully implemented in humanoid robots.

The most common strategy, nowadays, and based on dynamic walking, are the zero moment point (ZMP) (Kim et al., 2007) and the contact wrench sum (CWS) (Hirukawa et al., 2007). These techniques, which main principle is to cancel the total inertial forces actuating on the humanoid with the floor reaction force, are implemented in some of the most famous researcher humanoid robots like QRIO from Sony, ASIMO from Honda or HRP-2/HRP-3 from Kawada, allowing them to walk on uneven terrain and inclined plans, to run and to climb stairs. Another control strategy is based on biologically realistic walking (Popovic and Herr, 2004) and on the principle of spin angular momentum regulation. Recently, a dynamic balancing strategy control has also been successfully applied to the Dexter humanoid robot from Anybots. In this case and as opposite of ZMP strategy, it does not need preprogrammed footprints, being able to walk like a human and even to jump. Many other strategies have been studied, one last for instance is the passive-dynamic walking (PDW) that requires no external control or energy input, being the movement governed by the natural swinging of the legs (Collins et al., 2005; Asano and Luo, 2007).

In terms of current commercially available humanoid robots (see Figure 2.2), they are still designed to perform motions using open-loop control providing the users a simple paradigm to create pre-orchestrated multi-DOF walking gaits. These robots are usually not able to move on uneven terrain and it is difficult or impossible to get them to perform movements that require instantaneous reaction to momentary instability. A popular way to compensate for these predicaments is to over-capacitate servo torques and to incorporate large foot soles, low center-of-mass and better shock absorption, resulting in humanoid robots with little resemblance to the human physique, just as RoboSapiens from Wow Wee Toys (Figure 1.1b).

1.2 Purposes of this thesis

This thesis was developed in collaboration with Robosavvy Ltd and boosted the creation of the Humanoid Robotics Laboratory (Figure 1.2) of IDMEC-Center of Intelligent Systems, at Instituto

(a) Asimo from Honda (b) RoboSapiens from Wow Wee Toys

Figure 1.1: The humanoid robot from Honda and a commercial humanoid toy from Wow Wee Toys

Superior Técnico (http://humanoids.dem.ist.utl.pt/).

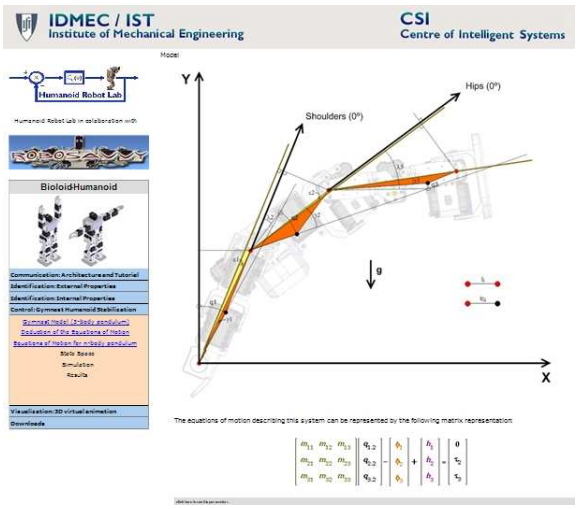


Figure 1.2: The actual web page from Humanoid Robotics Laboratory

The long term objectives of this thesis are to allow affordable commercial humanoid robots to walk, run, skateboard, jump and in general to react in a human-like physical way in dynamically unstable situations and uneven terrain. These goals can be achieved by applying closed-loop control techniques to the humanoid robot servos. The input data stream should consist of a multitude of sensors including servo position and torque, acceleration and inertial moment. The closed-loop control cycle should actuate the servos at rates of, at least 50Hz, which would give good responsiveness in a dynamic environment.

This thesis, however, has a more humble goal since it initiates the study of humanoids. Therefore, it is necessary to prepare the necessary conditions before achieving the desired long term goals:

1. The selection of a humanoid with high-torque servos.

2. The establishment of a real-time protocol communication between the PC, using Matlab/Simulink® Real-Time Workshop and the robot, for acquiring and sending data to the servos-actuators.
3. The identification of the physical and mechanical properties of the humanoid robot.
4. The identification and study of the behavior and responses of the high-torque servos.

After achieving this, then it is possible to create simulators incorporating all the information regarding the identification and behavior of the humanoid robot, aiming the control of the humanoid while doing complex tasks such as walking or skating.

For this project a simple simulator was created which shows the viability of having a particular control situation. The humanoid is seen as an underactuated three links robot hanging on a high bar, trying to mimic a gymnast human doing the hand-stand like the three link gymnast robot in (Takashima, 1989). The control strategy used is the well-known Linear Quadratic Regulator (Lancaster and Rodman, 1995; Ayala Botto, 2006), with torque being the servos input. Since the humanoid robot studied in this thesis has servo-actuators with speed as input, a relation between velocity and torque were established in order to be able to control it with torque.

1.3 Control solutions for stabilizing underactuated robots

Many studies were done with underactuated robots (Lee and Coverstone-Carroll, 1998; Aurelie et al., 2006). These robots are generally composed of two or three links in which the first joint is not actuated, or passive, whereas the others are actuated, or active (Figure 1.3). The objective is to swing up the robot from the vertical stable position to the upside-down position and then maintaining its unstable pose.

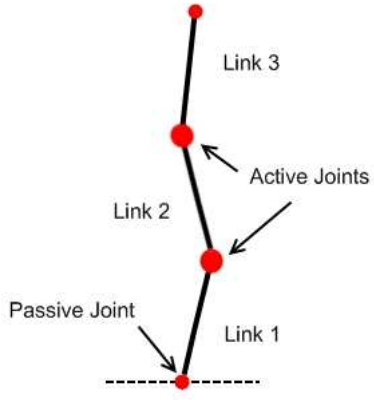


Figure 1.3: A 3-link underactuated robot in the upside-down region

Stabilizing the humanoid in his upright position is a challenging task and requires non-cancelation control techniques since the system is non-minimum phase. The most common controller used

as a first attempt is the Linear Quadratic Regulator (LQR) (Spong, 1994; Lee and Coverstone-Carroll, 1998). In this type of controller, a vector of constant gains (LQR gains or Kalman gains) are applied to the vector state of the system, obtaining new torque inputs for the system in order to stabilized it.

Another popular technique uses Partial Feedback Linearization to swing up a two-link underactuated robot (Spong and Block, 1995) and to stabilized it (Lee and Coverstone-Carroll, 1998). This control strategy derives from fully actuated robots, with no passive joints. For this type of underactuated robots, the model can be fully feedback linearized by a nonlinear feedback law (Spong and Vidyasagar, 1989). However, for underactuated robots, that is only true for the actuated joints (Partial Feedback Linearization). The rest of the dynamics of the system would still remain nonlinear. The solution found by (Spong, 1994) was to introduce a new condition called, Strong Inertial Coupling, to linearize also the dynamics corresponding to the passive joints. In this way, it was possible to feedback linearize an underactuated planar robot.

Partial Feedback Linearization demands a full knowledge of the model. Aiming a more robust stabilization technique that could handle model uncertainty, other techniques, like for instance, the nonlinear Sliding Mode Control (Utkin, 1992) technique was took in consideration for stabilizing a two-link underactuated robot in his upright unstable position (Lee and Coverstone-Carroll, 1998) and for the both phases, swing-up and stabilization (Qian et al., 2007). In this strategy, an additional term responsible for handling the uncertainties of the model is summed to a feedback linearizing controller improving the overall robustness of the controller.

Intelligent control has also been used to control underactuated robots. Among other strategies, an intelligent adaptive fuzzy radial Gaussian neural networks system for stabilizing a two-link underactuated robot in vertical unstable position (Qian et al., 2006) demonstrated to be globally stable, while an adaptive GA-tuning fuzzy PID control scheme, for the swing-up and stabilization of the same underactuated robot, has been implemented with some successfully results (Wu et al., 2007).

1.4 Contributions of this thesis

This thesis aimed for the identification of the internal and external parameters of the chosen humanoid, the Bioloid humanoid from Robotis.com®, and its application in a control situation using simulation where the robot should be able to do the hand-stand on a high bar. Therefore, some contributions were made within the development of the thesis, namely:

1. The construction of a serial protocol communication using Matlab/Simulink® and the humanoid robot.

2. The mass and inertia tensor calculation of every single component of the humanoid.

3. The analysis and identification of the internal behavior of the servo-actuators.

4. The development of a simulator for the humanoid doing a handstand on bar using virtual reality as animation.

5. The two sets of the humanoid 3D CAD drawing and its constituents. One set is detailed, resembling the reality pieces for mechanical analysis, while a less detailed one with precise real measurements is used in virtual reality animation.

6. The deduction of the equations of motions for a n-link inverted pendulum and its linearization along the vertical unstable position, and ways to control a n-link inverted pendulum with eccentrics masses.

1.5 Structure of the thesis

The rest of the thesis is organized in the following way.

Chapter 2 describes the serial protocol communication between the PC and the humanoid robot using Matlab/Simulink® as a platform.

In chapter 3, the identification of the external mechanical properties of the humanoid robot along with the internal properties of the servos is presented.

The construction of a simulator using virtual reality 3D humanoid as animation is shown in chapter 4.

In chapter 5, the equations of motion for the humanoid robot in his 3-link underactuated inverted pendulum configuration as well as for the generic n-link planar robot are presented. The state space model of the humanoid robot is also deduced.

Control, implementation and simulation results are presented in Chapter 6. Special emphasis is given to the Linear Quadratic Regulator (LQR) technique used to control the humanoid on a high bar doing a hand-stand.

Finally, in chapter 7, some conclusions and new ideas for future development are presented, pointing to the tasks left to be done in the future.

Chapter 2

Development environment set up

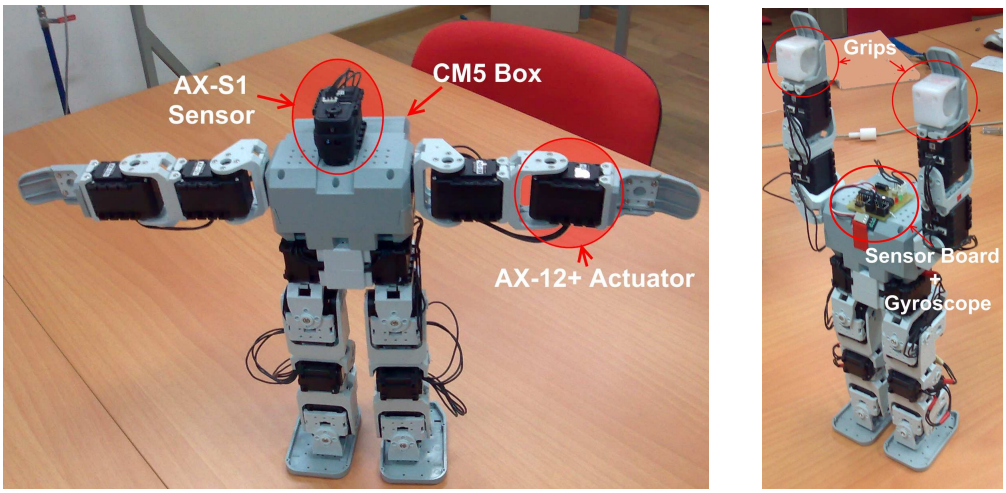
This chapter describes the hardware and software set up used along this thesis, that enabled the real time communication between the PC using Matlab/Simulink® and the humanoid robot.

Our development environment includes Simulink running on a Windows PC. Simulink can compile and offload control algorithms to various real-time hardware systems. The control loop is able to run both on-board the humanoid and on the PC. In both scenarios the sensory data and servo actuation commands should be streamed back and forth with the humanoid servos.

2.1 Humanoid robot

Given the above requirements, it was looked for a humanoid robot for our development environment. It was finally selected the Bioloid (Figure 2.1a) from Korean manufacturer Robotis.com. This was the humanoid kit of choice due to its well designed servo controllers that provide current, voltage, position and temperature sensing. It has a well documented open controller board and a well documented servo control protocol. Other humanoids platforms previously considered included: (a) KHR-1HV / KHR-2HV / Manoi / Robonova (Figure 2.2)- these are affordable humanoid kits whose servos provide position and current sensing. Their weight to torque ratio is probably better suited for running and skateboarding than Bioloid. However, documentation was lacking at the time this option was evaluated. (b) Custom Humanoid - many of the RoboCup teams and robot researchers build their own humanoid model using Aluminum and Fibreglass brackets and high-torque RC servos. A popular choice is the Robotis high-power RX and DX servos to actuate the robot.

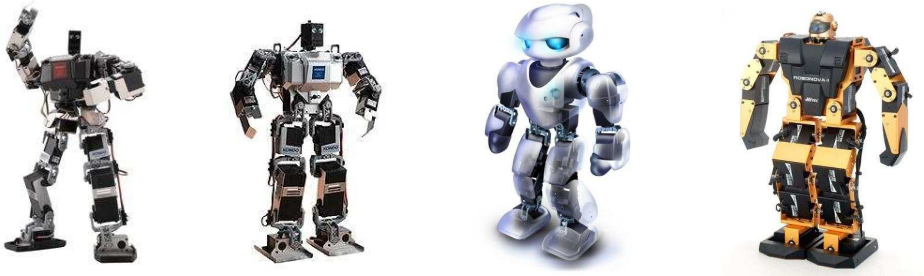
Two different configurations of the Bioloid humanoid were studied (Walking humanoid and Gymnastic humanoid). The humanoid showed in Figure 2.1a is the original humanoid configuration



(a) Walking humanoid configuration

(b) Gymnastic humanoid configuration

Figure 2.1: Humanoid in two different configurations and its main elements



(a) KHR-1HV (b) KHR-2HV (c) Manoi (d) Robonova

Figure 2.2: Current humanoid commercial platforms considered in the project

from Robotis set kit Bioloid, whereas the second robot (Figure 2.1b) is a modified version of that one. In this last one, the humanoid assumes the configuration of a human gymnast hanging on a bar. The resemblance is on purpose since it is our objective to have the robot doing a hand-stand on a high bar. Consequently, it was necessary to equip this robot with two additional hand grips, in order to allow it to hang on a bar, as well as a gyroscope plugged to a specified sensor board. This board replaced the custom head of the humanoid.

The humanoid Robot has 18 degrees of freedom (DOF) powered by DC servos. The "brain" of the robot is the CM5 board and it is located in the back of the humanoid. In the board, the microprocessor Atmega128 is responsible to send and receive information from the servos trough serial protocol communication RS485 and to send and receive data from the PC through the RS232 serial port.

AX-12+ Servo-actuator

The servos-actuators are the 'muscles' of the humanoid. These in particular (Figure 2.3) have some special features such as:

- precision DC motor and a control circuitry with networking functionality.
- 1 MBps communication speed.
- Full feedback on Position (300°), Speed, DC current, Voltage and Temperature.
- Voltage, DC current and Temperature automatic shutdown.
- Can be set as an endless wheel.
- High Torque servos.

Figure 2.3: AX-12+ servo-actuator

AX-S1 Sensor

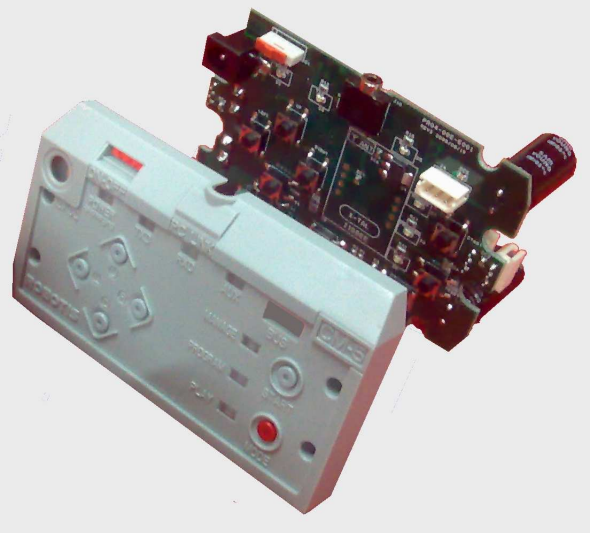
This sensor, resembling a servo, can detect distance, brightness, heat and sounds.

CM5 Box

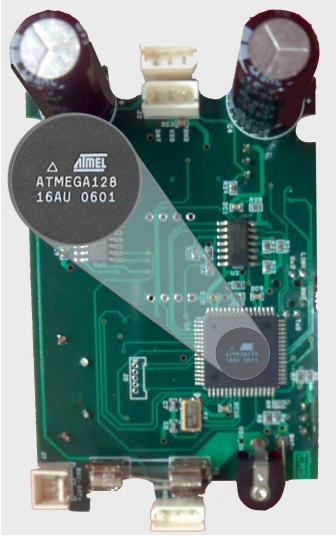
The CM5 (Figure 2.4) is the main controller of the humanoid. It consists of a microprocessor Atmega128, that can receive/transmite data to the servos and to the PC.

Sensor board and gyroscope

This sensor board (Figure 2.5) allow different types of sensors to be plugged into this board. In our case, the gyroscope KRG3 from Kondo was used. This gyro has 556 steps of resolution but without directly correspondence between velocity and the output encoders.



(a) CM5 box and its board



(b) CM5 microprocessor Atmega128

Figure 2.4: CM5 box, CM5 board and Atmega128

Figure 2.5: Pepper sensor board and KRG3 gyroscope from Kondo

2.2 Hardware architecture

Figure 2.6 shows the existing humanoid architecture and the control architecture used. The humanoid controller named CM5 is connected to the controllers of the servos through a RS485 bus. The usual approach to teach the robot is to use a humanoid proprietary software that connects to a PC through the RS232 serial line. The CM5 has however an Atmega128 microcontroller (Figure 2.4) with a bootloader which allows users to change the code and access directly to the servo controller parameters. A small program was developed in the CM5 controller to implement a protocol for transmitting/receiving through the serial port the data from/to the servos.

Note: The servos communicate through asynchronous serial communication with 8 bit, 1 stop bit and no parity.

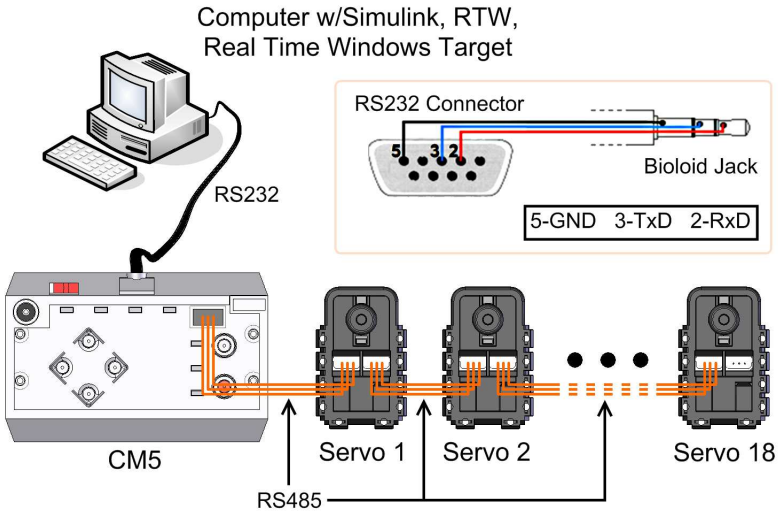


Figure 2.6: Hardware architecture diagram

2.2.1 Protocol communication

Servos are wiling to receive/send package information from/to the CM5. Therefore, data instructions are sent to the servos and status packets are received from them.

Instruction packet

In order to give to a specific servo a specific command, an instruction packet is transmitted to it with the following structure (Figure 2.7).

Figure 2.7: Instruction packet

Where:

1. The first two bytes (0xFF 0xFF) means that a new instruction is going to be sent.
2. The third byte says which servo will receive the instruction. When the ID of the servo is set to 0xFE, then the instruction is sent to all servos.
3. The fourth byte represents the length of the instruction packet, i.e. the number of bytes of the packet length without counting the first three bytes.
4. The fifth byte represents the instruction that the servo should execute. These can be:

   0x01 Ping: Used for obtaining a status packet.

0x02 Read Data: Used to read values of one servo, such as its current position.

0x03 Write Data: Similar to Read Date, but used to write instead (e.g. send a new goal position).

0x04 Reg Write: Similar to Write Data, but stays in standby mode until the Action instruction is given.

0x05 Action: Triggers the action registered by the Reg Write instruction.

0x06 Reset: Reset all the parameters of the servos to its original values.

0x83 Sync Action: Used for control servos at the same time.

5. The sixth byte stands for values to be sent along with the instruction (e.g. the value of the goal position).

6. The last byte gives the checksum of the packet.

Status packet

After sending an instruction packet a status packet is sent back to the main controller (CM5) (Figure 2.8).

Figure 2.8: Status packect

The status packet is similar to the instruction packet, with the difference of having a byte error instead of the instruction command. When the error byte is 0x00, then the instruction packet went well, otherwise one or more errors occurred (table 2.1).

Bit	Error
7	0
6	Instruction Error
5	Overload Error
4	Checksum Error
3	Range Error
2	Overheating Error
1	Angle Limit Error
0	Input Voltage Error

Table 2.1: Status packet errors

Read and write instructions

Read and write commands are defined as parameters in the instruction packet and they are different depending they are sending or requesting an information (table 2.2).

When writing information to a servo, the first parameter byte stands for the command the servo should undertake (e.g., Goal Position is 0x1E, and Moving Speed is 0x20). The remained of the parameter bytes represent the value of the command.

Reading data is similar to writing, but the parameters bytes are always two, where the second byte tells the number of bytes to be read from the first parameter byte.

Command	Address	Access
CW Angle	0x08 (8)	Write
CCW Angle	0x08 (10)	Write
Goal Position	0x1E (30)	Write
Moving Speed	0x20 (32)	Write
Present Position	0x25 (36)	Read
Present Speed	0x27 (38)	Read
Present Load	0x29 (40)	Read
Present Voltage	0x2A (42)	Read
Present Temperature	0x2B (43)	Read

Table 2.2: Project read-write instructions

Endless turn mode

Endless turn mode, e.g. the servo behaves like a continuous rotating wheel, can be set by turn CW and CCW to zero.

Goal Position

The servos have 10 bit resolution in terms of position corresponding to 300°. However exists an invalid zone that the servo can not reach. When the servo is set to endless turn mode, the sensor is not able to read the position of the invalid zone. At 150°, the servos are in its middle position. For instance, the physical configurations of the Humanoids of Figure 2.1(a,b) have all the servos in that position.

Moving speed

Moving speed can be set in 1024 increments.

Present position

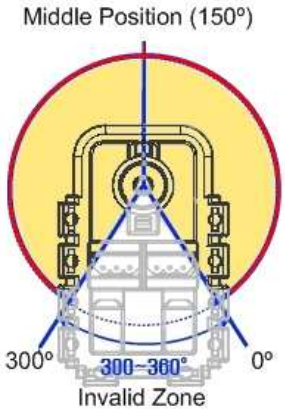


Figure 2.9: Status Packect

Reads the actual position of a servo (10 bits resolution for 300°).

Present speed

Reads the actual speed of a servo (10 bits resolution). No information about the units used.

Present load

Reads the actual DC current consumed by the servo (9 bits resolution with 10th being the load direction). No information about the units used.

Present voltage

Reads the actual Voltage of the battery (8 bits resolution or 0.1 Volts resolution).

Present temperature

Reads the actual Temperature of the servo (8 bits resolution or 1° Celsius resolution).

2.2.2 Programming the micro-controller Atmega128

The next step, was to build a small C program to run on the microprocessor Atmega128 in order to prepare the communications with the PC.

The C program was developed from an existent C code for the microprocessor inside the CD that comes with the humanoid robot, the "example.c". The changes undertaken inside this code were only done inside the main function. In the piece of code of Figure 2.10 a simple situation is shown (starts at *code to be inserted*) in were the CM5 starts by reading two bytes from the PC side corresponding to the final desired position for servo 1. Afterwards, this desired position is sent to the servo. The cycle ends by sending the actual position of the servo back to the PC.

The code was compiled by using the *notepad programmers 2* from WinAVR using the command *Tools => [WinAVR] Make All* . A .hex code is finally generated, and this code is uploated to the Atmega128.

```
(. . .)
int main(void) {
  byte bCount,bID, bTxPacketLength,bRxPacketLength;

  PortInitialize(); //Port In/Out Direction Definition
  RS485_RXD; //Set RS485 Direction to Input State.

  //RS485 Initializing(RxInterrupt)
  SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);

  //RS232 Initializing(None Interrupt)
  SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0);

  gbRxBufferReadPointer = gbRxBufferWritePointer = 0;  //RS485 RxBuffer Clearing.

  sei();  //Enable Interrupt -- Compiler Function

/***************************************
 *         code to be inserted         *
 ***************************************/

  //servo to be used
  bID = 1;

  //goal position splited in lo and hi byte
  byte lo, hi;

  while(1)
  {

    //read from MATLAB the goal position
    lo=RxD8();
    hi=RxD8();

    //send the goal position to the servo
    gbpParameter[0] = P_GOAL_POSITION_L;     //Address of Firmware Version
    gbpParameter[1] = lo;                    //Writing Data P_GOAL_POSITION_L
    gbpParameter[2] = hi;                    //Writing Data P_GOAL_POSITION_H
    gbpParameter[3] = 0xff;                  //Writing Data P_GOAL_SPEED_L
    gbpParameter[4] = 0x3;                   //Writing Data P_GOAL_SPEED_H
    bTxPacketLength = TxPacket(bID,INST_WRITE,5);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);

    //get present position
    gbpParameter[0] = P_PRESENT_POSITION_L; //Address 36 of present position LSB
    gbpParameter[1] = 2;                    // Read 2 bytes starting at address 36
    TxPacket(bID,INST_READ,2);              // 2 means 2 bytes in parameter array
    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);

    //read a char
    RxD8();

    //send back to MATLAB the actual position of the servo
    TxD81(gbpRxBuffer[5]);
    TxD81(gbpRxBuffer[6]);
  }

/***************************
 *         end code        *
 ***************************/

}
(. . .)
```

Figure 2.10: main function of the C code to be uploaded to the CM5

In a similar way, the code can be changed to receive only desired speed and get other parameters such as speed, dc current, voltage and temperature.

After this, it is necessary to upload the program to the microprocessor. For that purpose, the *robot terminal* that comes with the Humanoid was used. After verifying that the serial cable is plugged to the PC and CM5, the CM5 is turned on. In *Setup => Connect* verify if the CM5 is connected at 57600bps and if the used port is the COM1. Now enter in the boot loader by pressing

in the same time the red button of the CM5 and #. The screen of Figure 2.11 must appear.

Figure 2.11: Bootloader Screen

Write now the command *Load*. Press *Enter*, then go to *Files => Transmit file* and open the .hex code. Finally go to *Files => Disconnect*.

2.3 Software architecture

The device drivers to send/receive data to/from the servos using the defined protocol were developed in Simulink / Real Time Windows Target. For doing this, a C program for Atmega128 was written (see section 2.2.2). Hereafter, it was necessary to establish a protocol for the serial communication between the PC and the CM-5. Finally, a C-MEX S-function to communicate with the CM-5 throughout UART (universal asynchronous receiver / transmitter) for completing the serial communication bridge was written in C. This architecture was implemented as it transparently maps Simulink variables into the servos motion. There is now a way to identify the parameters of the humanoid models making online experiments, as Matlab/Simulink® is a unique tool, widely used, for system identification and control.

In order to guarantee data samples at precise time inputs sent to the servos, a Simulink/RTWT implementation was used. The Real-Time Windows Target is a real-time kernel which permits C code, generated and compiled by the Real-Time Workshop from Simulink block diagram models, to run in real time, at ring zero, under the Windows operative system.

2.3.1 C-MEX S-function

A protocol for the serial communication between the PC and the CM-5 had to be established. A C-MEX S-function written in C was created to communicate with the CM-5 throughout UART (universal asynchronous receiver / transmitter). This code is meant to finish the protocol bridge started by the C code for the CM5. Hence, its main goal is to sent, at each sample time, a desire final position to a servo through CM5, receiving back its actual position. For full understanding of the S-function code it was splited into 5 simple parts in Appendix A.

2.3.2 Simulink block diagram

Finally, the construction of a Simulink diagram block is necessary to run the protocol communications. A small example is described bellow.

1. Open Matlab and then Simulink.
2. Build the block diagram shown in Figure 2.12.

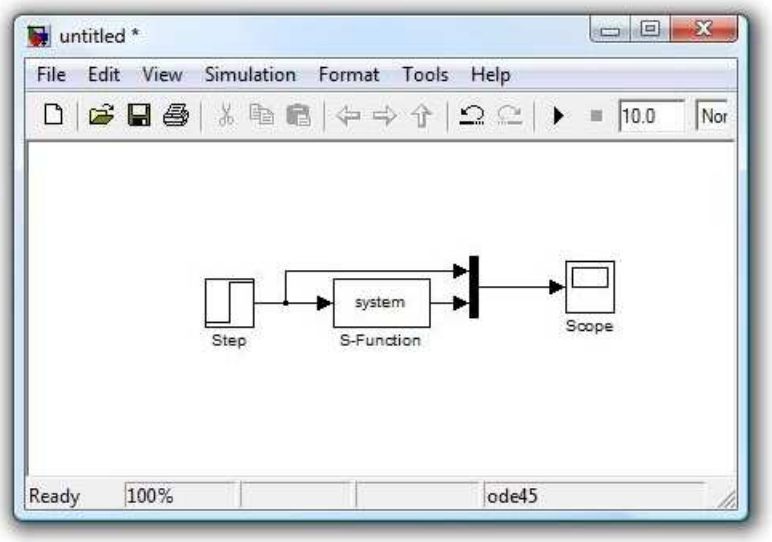


Figure 2.12: Simulink block diagram

3. Double click in the *Step* block and then change its parameters as follows: Step Time: 5; Initial Value: 300; Final Value: 800; Sample Time: 0.01.
4. In *Simulation > configuration Parameters » Solver*, chose fixed-step and then 0.01 seconds as the sample time.
5. In *Simulation* select *external mode*. This will allow to run Simulink in real time.
6. Save the block diagram in the same directory as the S-function code.
7. Double click in the S-function block to open *Function Block Parameters: S-Function*.
8. In the S-function name put the name of your S-function name and press *Apply* and then *Ok*.
9. Press Incremental build button to build the model and wait until the play button becomes enable (▷).
10. Before testing the application, verify if the CM5 is plugged to the PC and if it is OFF.
11. Attach the servo number one to the CM5 (other can be chosen but it has to be declared it in the MCU code).

12. Finally, turn on the CM5 and then press the play button. Something similar to the next graph (Figure 2.13) must be seen.

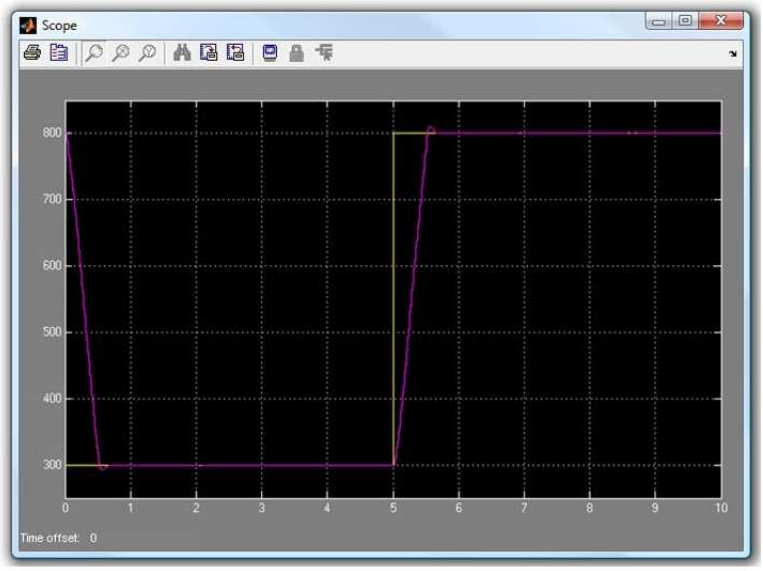


Figure 2.13: Real-time servo step response

2.4 Real time issues and communications delays

A program in C++ was built to measure all the current positions of the servos and therefore to verify the real frequency of the application. In this last case, 19 real servos were used. The servo position is a 10 bit value that has to be divided in a high and low bit in order to be sent throughout the RS232 to the PC. Hence, the theoretical value, without latency, is given by equation (2.1).

$$f = \frac{57600}{(19+1) \times 2 \times 10} = 144 \ \text{Hz} \tag{2.1}$$

Where:

- 57600 bps is the Baud Rate.
- 19+1 means the information sent by 19 servos plus 1 additional value used for resynchronization.
- 2 stands for 2 bytes (the encoder number of each servo is 10 bits of resolution).
- 10 is the number of bits (8 bits for the data +1 parity bit +1 stop bit)

The measured frequency, 143.4 Hz, was not far from the theoretical value, being just 0.6Hz bellow. This means that the latency is responsible for just 0.42% delay of the all process. Testing

the latency when a given order is sent by the PC, read it and send it back to the PC by the CM5 takes about 1.29ms, as shown in Figure 2.14 after a running test of 1000 samples.

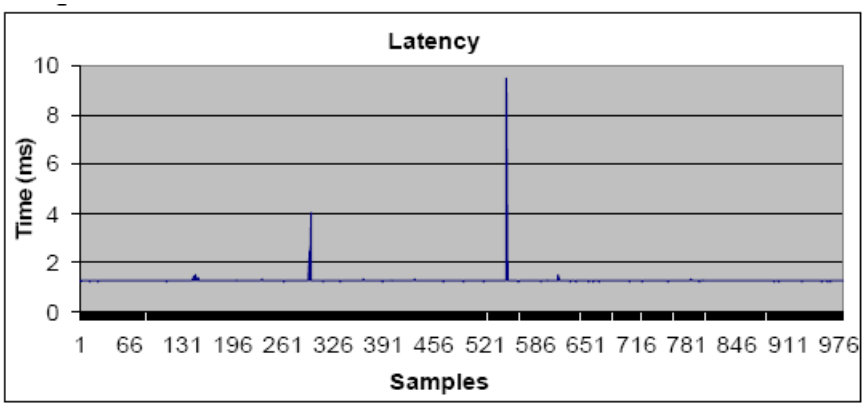


Figure 2.14: Communications delays from PC and CM5

Thus, it was decided to use a sample rate of 0.01 seconds (100 Hz) in all future tests.

Chapter 3

Humanoid Identification

In order to capture the static and the dynamic properties of the humanoid robot, both the mechanical properties of all its components, such as mass and inertia, as well as its servos dynamic responses, must be known to a certain degree of accuracy. These dynamic properties will be used in Simulink® and SimMechanics® in order to get an accurate simulator for the real humanoid robot aiming at a good control strategy.

3.1 Mechanical and physical properties identification

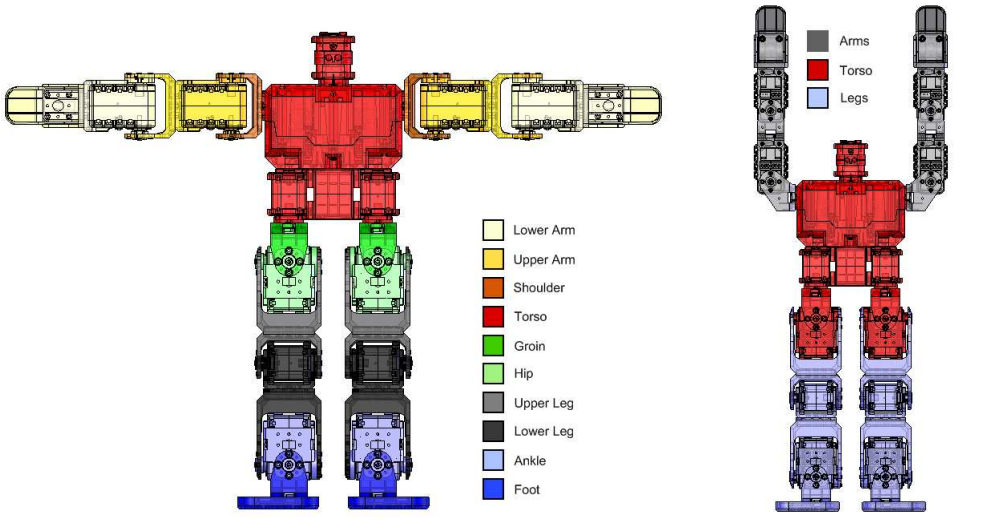
An accurate static model of the Humanoid Robot can be obtained based on the physical properties of their components. Typically, by knowing the mass, center of mass and the inertia tensor of each element of the robot it is possible to get a quite reliable model that can be further used in simulation and control. For quantifying the masses of each element, a precision scale with a resolution of 0.5×10^{-4} Kg was used. The centroid of each mass was then found by using the SolidWorks® software package, after the detailed elements of all the pieces involved were drawn in this 3D CAD software. It was assumed here that, except for the servos, all the pieces are of isotropic nature. A simple experiment has shown that the maximum error obtained for the geometric position of the servos' centroid is of 0.5 mm on each Cartesian direction. Finally, the inertia tensor of each element was determined through the SolidWorks® software.

In this thesis, two different configurations were studied (Walking Humanoid and Gymnast Humanoid). In the first configuration (from fabric), the humanoid can make use of all of its servos, and therefore, is able to walk, jump, run, skating and so on. In order to accomplish this, the robot can be seen as an assembly of 19 main blocks (Figure 3.1b) connected and powered by 18 smart servo-actuators working as joints (Figure 3.2a). In the second configuration, the humanoid was rebuilt in order to resemble a gymnast exercising on a high bar with that purpose. Designed like

this, the robot can be split into 3 main blocks, arms, torso and legs (Figure 3.1b) powered by 4 servo-actuators, two for the shoulders and two for the hips (Figure 3.2b).

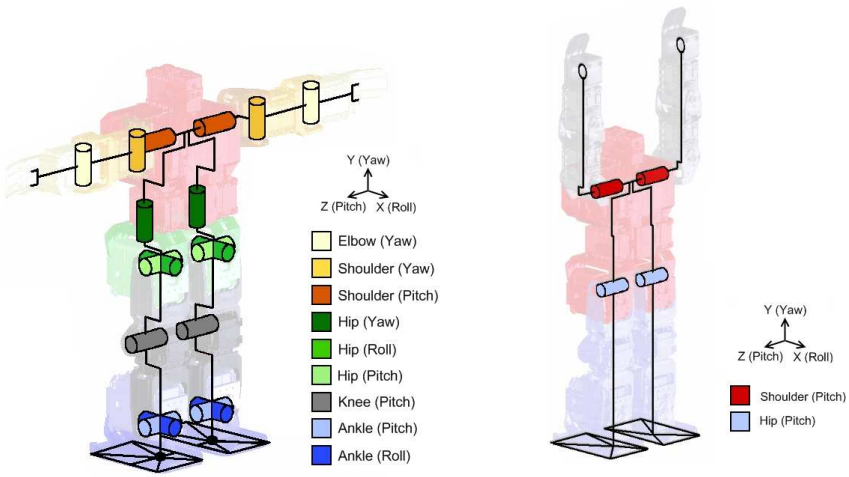
Another important feature to check is the structure and the disposition of the servos (joints) in each humanoid configuration. This would give an idea of the real mobility of each humanoid (Figure 3.2(a,b). In Figure 3.2b, however, only the main joints of the humanoid are presented in its gymnast configuration, since it does not make use of the others servos.

The masses, tensor of inertias and drawings of the main blocks of each configuration are given in Appendix C and D. These drawings also contains the position of the center of gravity of each body.



(a) Walking humanoid (b) Gymnastic humanoid

Figure 3.1: Walking and Gymnast humanoid main blocks



(a) Walking humanoid (b) Gymnastic humanoid

Figure 3.2: Walking and Gymnast humanoid joints and skeleton

3.2 Dynamic servo-actuators properties

The servo-actuators of the humanoid robot (Figure 3.3) are powered by a 10 Volts DC battery inside the CM5-Box through a three-wire daisy chain (Figure 3.4). From the data wire the final desired angular position and velocity can be sent to the servos, whereas a set of output signals can be retrieved in the same way (RS485), such as the actual servos angular position, angular velocity, DC current, temperature and voltage. These data are processed by the micro-controller Atmega8 inside each servo (Figure 3.5).

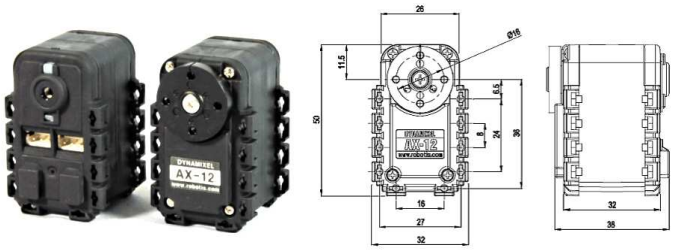


Figure 3.3: Servo Actuator AX-12 and drawings

1

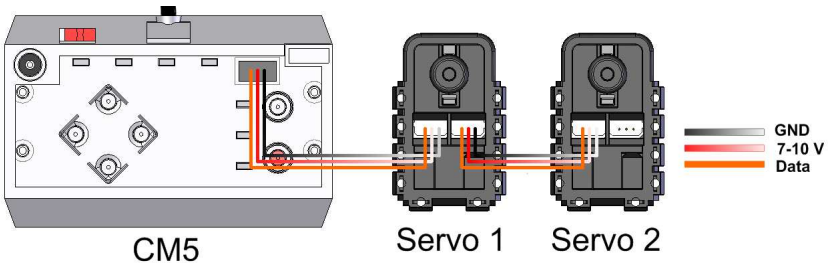


Figure 3.4: Daisy Chain Wiring

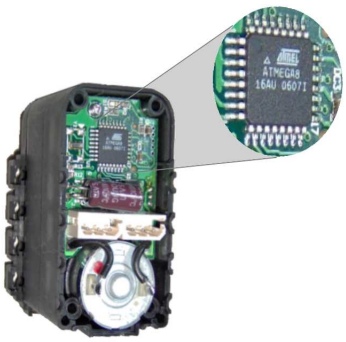


Figure 3.5: AX-12 microprocessor

3.2.1 DC motor

A common DC motor is a simple electromagnetism mechanism based on the Lorentz force law. It defends that any current-carrying wire placed within a magnetic field experiences a mechanical force which is proportional to the current and to the strength of the magnetic field and perpendicular to both, causing a torque. In a DC motor (Figure 3.6), wires (coil) and the flux density of the magnetic field (B) are arranged in order to develop a torque about the axis of the rotor (rotating part of the motor). To maintain a DC motor spinning, commutators are used to reverse the current every half a cycle, keeping in that way the torque in the same direction.

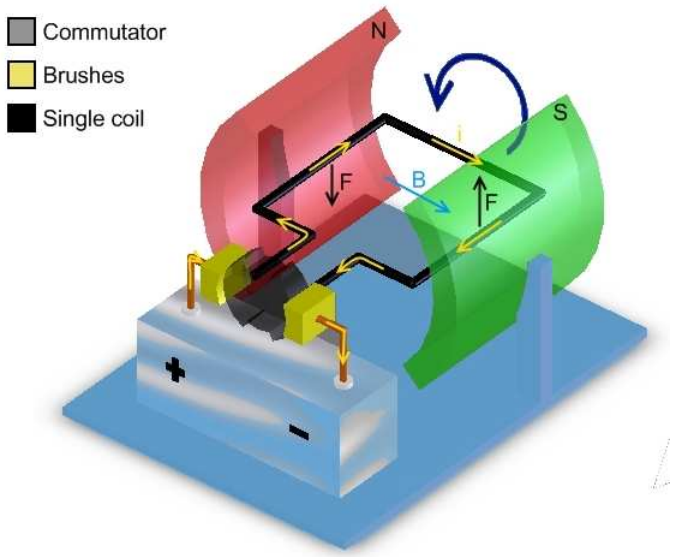


Figure 3.6: DC motor operation

3.2.2 Mathematical model

By analyzing the behavior of the servos it is possible to deduce their mathematical model. Remembering that a servo is a DC motor, the only source of energy that makes its motion possible is the current voltage of the battery. Therefore, and in order to be able to change the output angular velocity of the servo, one out of two solutions can be taken. The first one is to use resistors. The second one and much more effective in terms of dissipated energy is to use electronic control. The most common technique in this last case it to use a circuit known as a chopper to regulate the average voltage applied to the servo, and consequently the output velocity. To do this, the chopper circuit, which is made of thyristors or any other mercury arc rectifiers, turns on and off the supply voltage very rapidly. This technique is also known as pulse width modulation, best known as PWM and it is often controlled by a micro-processor. Figure 3.7 presents the electrical equivalent circuit of a simple DC motor using a simple chopper circuit.

where:

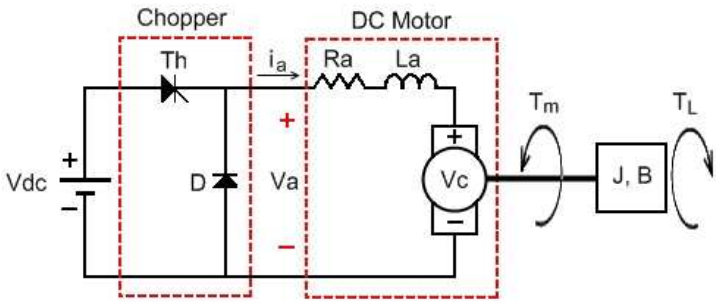


Figure 3.7: Electrical Representation of a DC motor

- V_{dc} is the voltage source.
- V_a is the average voltage supply.
- V_C is the induced voltage or the back or counter electromotive force (CEMF) which opposes the voltage supply, proportional to the motor speed (Figure 3.6).
- Th is a GTO thyristor which function is to behave like a gate for opening and closing the circuit.
- D is a free-wheeling diode, used in power switching applications.
- R_a is the resistance of the armature coil.
- L_a is the inductance of the armature coil.
- i_a is the current in the armature coil.
- J is the inertia of the rotor.
- B is the damping coefficient of the rotor.
- T_m is the electromagnetism torque.
- T_L is the mechanical load torque.

The Kirchoff's voltage law for the electrical loop and an energy balance on the mechanical part of the system, is:

$$\begin{cases} V_a - i_a R_a - L_a \frac{d}{dt} i_a - k_v \omega_a = 0 \\ T_m - B\omega_a - J \frac{d}{dt} \omega_a - T_L = 0 \end{cases} \tag{3.1}$$

where:

- $k_v \omega_a$ is the back EMF (V_c) which is proportional to the angular velocity of the rotor (w_a). k_v is determined by the flux density of the magnetic field.

Finally, it has been seen that the servo can receive a desired angular position and a desired angular velocity as input. When the servo receives the desired angular velocity, it converts into the necessary average voltage consumption and then into the correspondent PWM signal. This procedure enables the output angular velocity to be equal to the desired one when working without external load. Figure 3.8 provides a possible block diagram control of the servos.

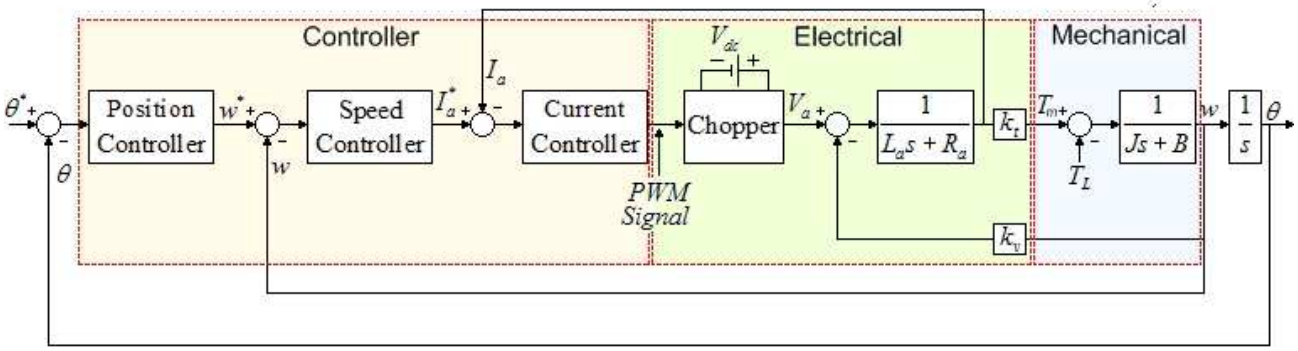


Figure 3.8: Possible internal block diagram control of the servos

Therefore, a set of tests were performed on the servos aiming the understanding of their behavior, using for that purpose, the code developed in Appendix B, witch sends a reference speed to a servo and receives from it a roll of different data, such as current angular position, angular velocity, DC current, voltage and temperature. Servo angular velocity is also estimated from current position using an online filter. The results are presented in the following subsections.

3.2.3 Close-loop position control

By default the servos are configured for position control. In fact, all servos have an internal feedback position control loop. This characteristic can be easily confirmed by the simple experiment shown in Figure 3.9. The servo tries to follow the desired time varying sinusoidal reference position by changing its actual D/C current (load) charge through time, even in the presence of an external torque applied at time instant t=6 sec.

This result shows that the block diagram of the Figure 3.8 seems to be correct.

3.2.4 Open-loop velocity control

After some tests, it was shown that the servos do not have internally any angular velocity feedback control. This can be experimentally confirmed by applying an external torque to the servo while in constant rotating velocity. From Figure 3.10 it can be concluded that the current consumption

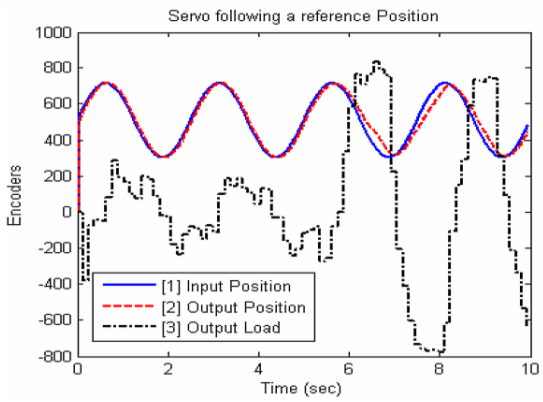


Figure 3.9: Close Loop Position Control

is not able to respond accordingly after the fifth second when a torque is applied, so the reference angular position cannot be followed.

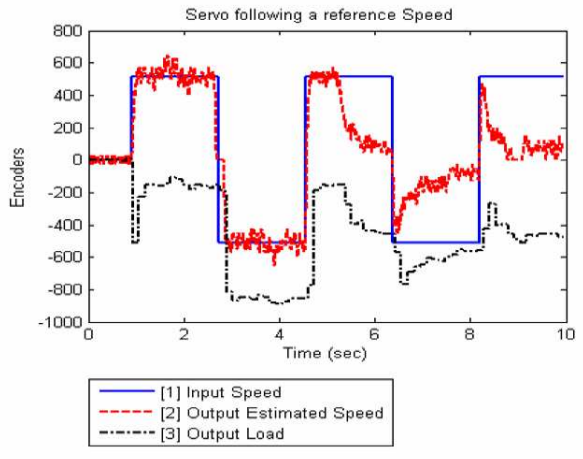


Figure 3.10: Open Loop Velocity Control

3.2.5 Stiction

Stiction is a physical phenomenon that is present in almost any system with moving components. Therefore, its characterization is essential for obtaining an accurate dynamic model of the servos. A simple way to quantify stiction can be made through the following experiment: starting with the servo rotating at a constant speed in one direction, progressively slowing it down until it stops, and then slowly increase its rotating speed in the opposite direction. With this experiment it should be possible to identify the typical dead-zone effect due to stiction. In our case this was clearly quantified to be around 7-10% of the full range when no load is applied to the servo, as can be seen in Figure 3.11.

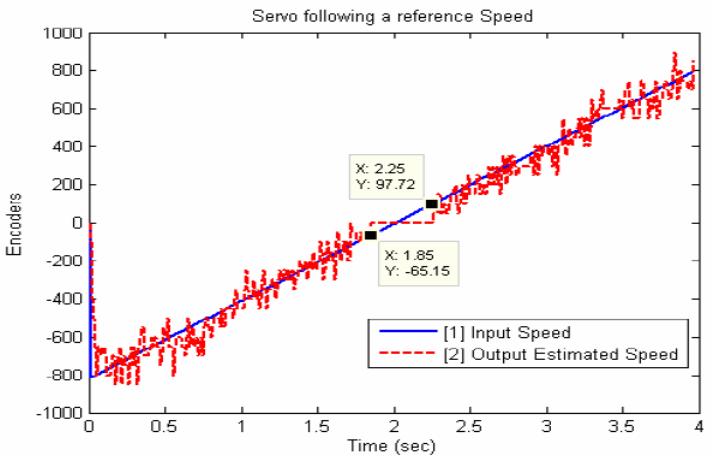


Figure 3.11: Stiction Dead Zone

3.2.6 Voltage

Another parameter with relevance to the behavior of the system is the voltage supplied to the servos. Experiments show that the output estimated velocity error is proportional to the voltage supplied to the servo. In fact, a good output velocity estimation is achieved only if the battery is charged around 10 V, as can be seen from Figure 3.12

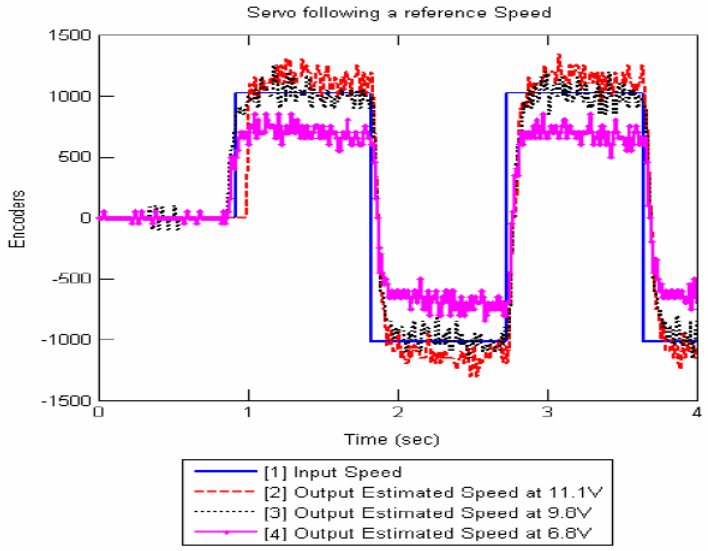


Figure 3.12: Effects of the supplied voltage to the servos in the outputs velocity response

3.2.7 Temperature

Temperature was the last parameter to be tested in order to check its influence in the behavior of the servos. During the tests, the temperature of the servos were within the interval of 25º to 40º.

At these conditions, no visible effect in the response of the servo was observed and therefore the effect of temperature is negligible.

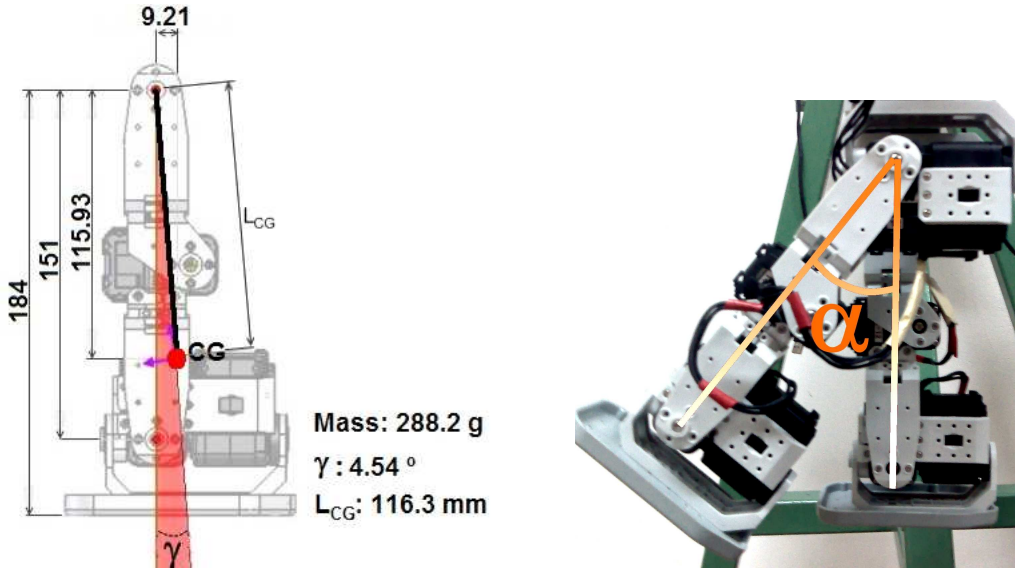
3.2.8 Load and Speed

Current load and speed are data possible to be retrieved from the servos, however, these data can only be sampled at 10 Hz, being far from the desired 100 Hz for the project. Furthermore, it was not possible to deduce a direct correspondence between the values of the encoders and the units of the international system of units (SI).

3.2.9 Torque as input signal

Torque is a very important variable to control a mechanical system. Therefore, the relation between torque and angular velocity when in free run mode (no directly control of the speed) has to be found. It has been seen that the angular velocity input signal is directly proportional to the servos's voltage consumption, and consequently to the current intensity. Since torque is also directly proportional to the current intensity, a proportional gain can be deduced from torque to angular velocity.

This gain was deduced making the servo lifting a well known mass, the humanoid leg (Figure 3.13a), for various input angular speed signals. Measuring for each case the correspondent torque by knowing the angle of the leg about the vertical (Figure 3.13b), it was possible to determine the correspondence between input speed signal and output torque (table 3.1) through equation (3.2).



(a) Humanoid Leg (b) Humanoid Leg Inclination

Figure 3.13: Speed to Angle Inclination of a Humanoid Leg

$$T_l = mgL_{CG}\sin(\alpha - \gamma) \tag{3.2}$$

Where:

- m is the mass of the humanoid leg (Figure 3.13a).
- g is the acceleration of gravity.
- L_{CG} is the length between the center of gravity of the leg and its rotational axis (Figure 3.13a).
- α (Figure 3.13b) is the angle of the humanoid leg about the vertical.
- γ (Figure 3.13a) is the angle between the humanoid leg and its center of gravity.

Arms		
Input Angular Speed (Enc)	Angular displacements of CG (deg)	Load Torque (Nm)
0	0	0
40	0	0
80	0	0
100	7.96	0.045
150	16.46	0.093
200	25.46	0.141
250	35.46	0.191
300	48.46	0.246
350	66.96	0.302

Table 3.1: Speed input signal to Torque correspondence

As observed in Figure 3.14 the relation between the output of the servo and the input angular speed is not linear, due to the dead zone of the servo (section 3.2.5). Nevertheless, it was obtained the real relation of Torque-Speed of the linear zones given in equation (3.3) when the nominal voltage is 9.8V.

$$T = \begin{cases} 1.023 \times 10^{-3}\omega - 60.61 \times 10^{-3} & \omega > 0 \\ 1.023 \times 10^{-3}\omega + 60.61 \times 10^{-3} & \omega < 0 \end{cases} \tag{3.3}$$

When the velocity is set to a maximum of 1023 encoders/sec, the output torque is about 0.99 Nm.

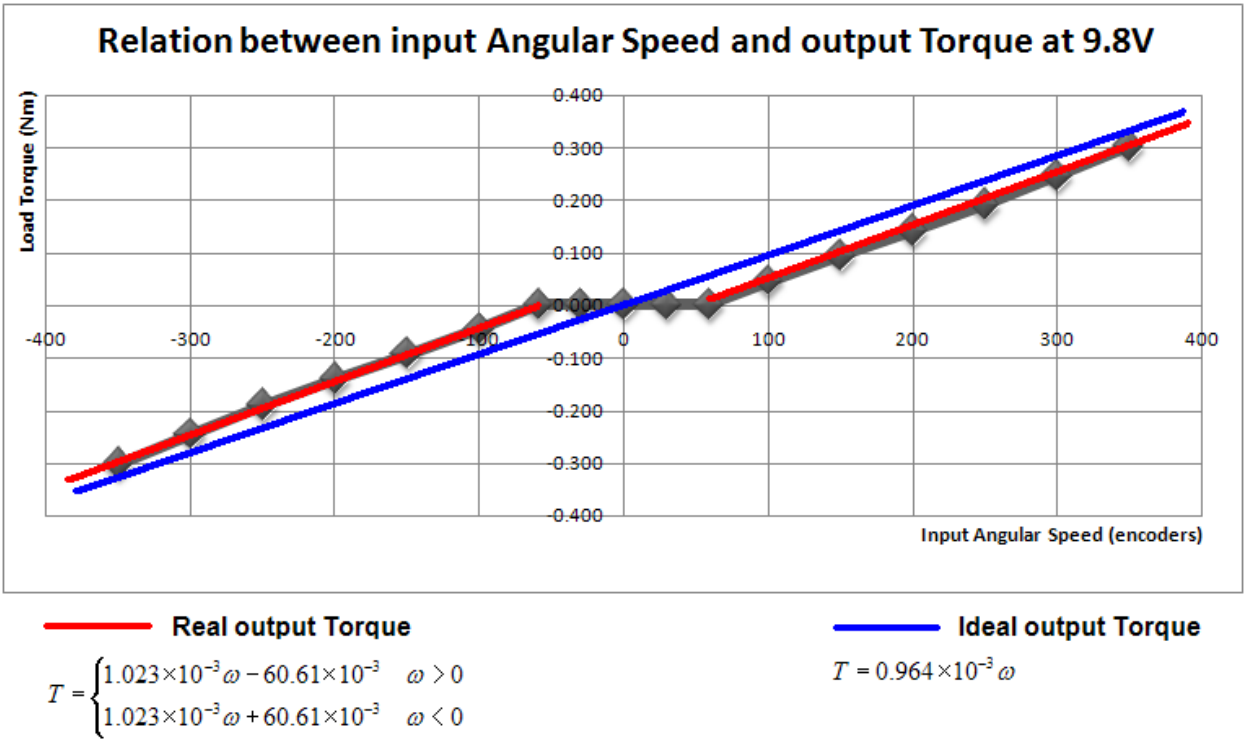


Figure 3.14: Speed to Torque Relation

3.3 Dynamic Servo Identification and Validation

For the identification of the dynamic behavior of the servos it was considered the relation between the reference input velocity and the correspondent estimated velocity obtained through the following equation:

$$\hat{\dot{\theta}}(t) = \frac{\theta(t) - \theta(t-1)}{T_s} \tag{3.4}$$

Where

- $\hat{\dot{\theta}}(t)$ is the estimated velocity at time instant t.
- $\dot{\theta}(t)$ is the angular velocity at time instant t.
- $\theta(t)$ is the the angular position at time instant t
- $\theta(t-1)$ is the the angular position in the previous time instant
- $T_s = 0.01$ sec. is the sampling period.

The classical prediction error method was used for the identification of the servo dynamic model, using the identification data shown in Figure 3.15.

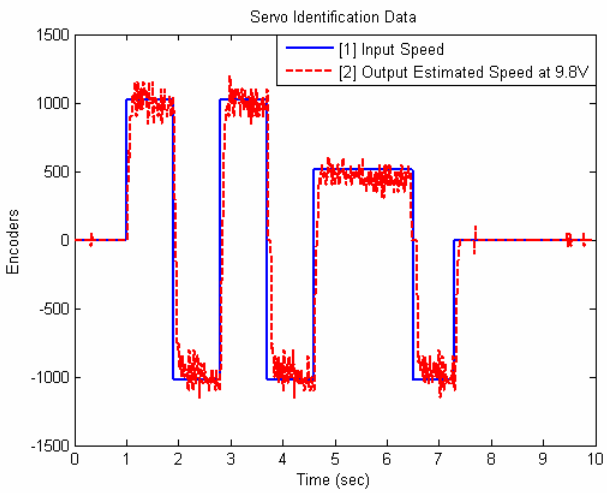


Figure 3.15: Servo Identification Data

After testing several tentative models with different orders, a Box-Jenkins(2,1,2,1) (Ljung, 1987) was found to best approximate the desired dynamical behavior of the servo. The Box-Jenkins model that results in the best data fit is the following:

$$TF = \frac{0.06217z}{z^2 - 1.469z + 0.5544} \tag{3.5}$$

Figure 3.16 compares the real output of the servo with the one estimated by the Box-Jenkins model for the validation data. It can be concluded that the dynamic characteristics of the servo are well captured by the Box-Jenkins model.

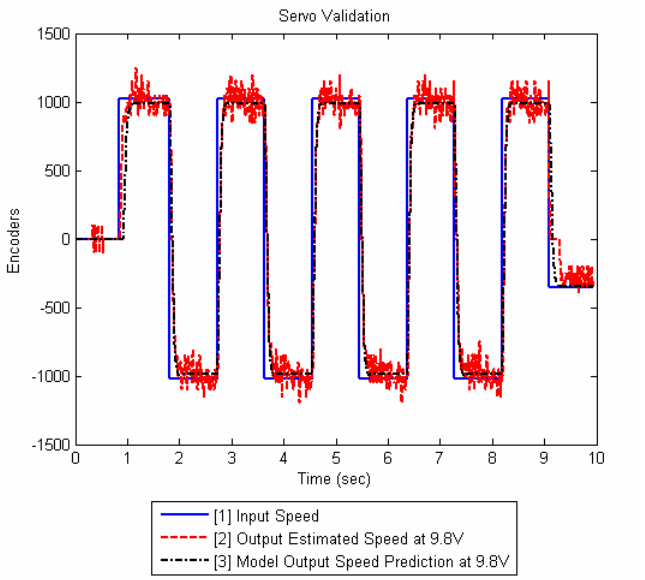


Figure 3.16: Servo Validation Data

Chapter 4

Simulator

This chapter presents the simulator of the Gymnastic humanoid hanging on a high bar. For this simulator another set of the humanoid pieces were drawn in SolidWorks2006®. These ones are simpler than the first set, with less details and with the purpose of being used in a virtual reality world. Nevertheless, these pieces not only allow the user to construct in virtual reality the Gymnastic humanoid but also the Walking humanoid as any other Bioloid robot configuration.

The simulator was constructed in Simulink/SimMechanics® with the capability of simulating friction between hands and the high bar, the gyroscope sensor and servo properties.

4.1 The humanoid model

The simulator has the purpose of simulating the behavior of the humanoid hanging on a high bar. Therefore, the exact mechanical properties of the humanoid, such as the real position of the centers of gravity and also the real measurements of its constituents, are needed. These properties were fully described in chapter 2 and implemented in the simulator (Figure 4.1 and tables 4.1 and 4.2).

	$l\ (mm)$	$lc\ (mm)$	$\epsilon\ (^o)$	$\gamma\ (^o)$
Link 1 (Arms)	143.6	68.7	2.29	2.94
Link 2 (Torso)	115.8	57.5	-0.49	17.65
Link 3 (Legs)	184.0	116.3	0.00	4.54

Table 4.1: Main measurements of the Gymnastic humanoid

The inertia tensor is taken at the center of gravity of each block and oriented with the output coordinated system. In our model the robot is facing left (minus x) when in the vertical unstable

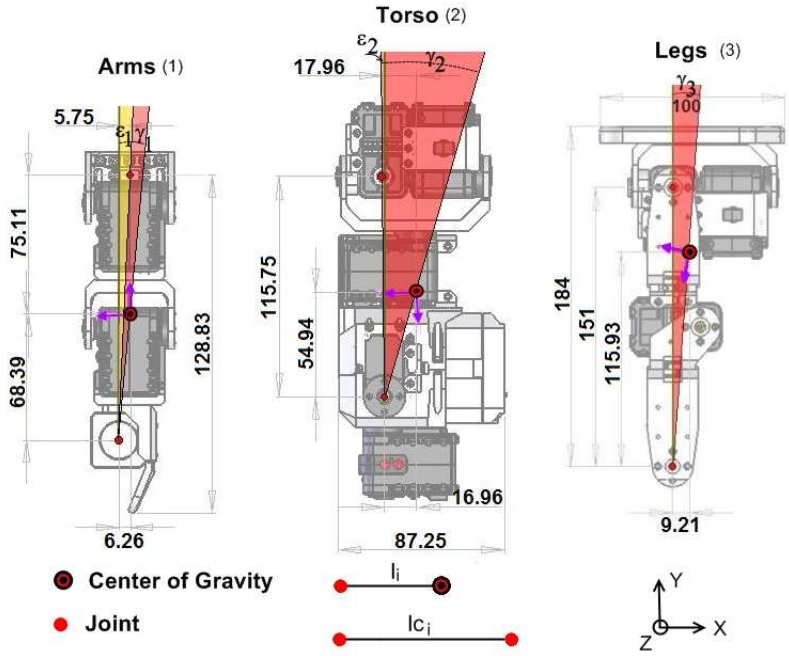
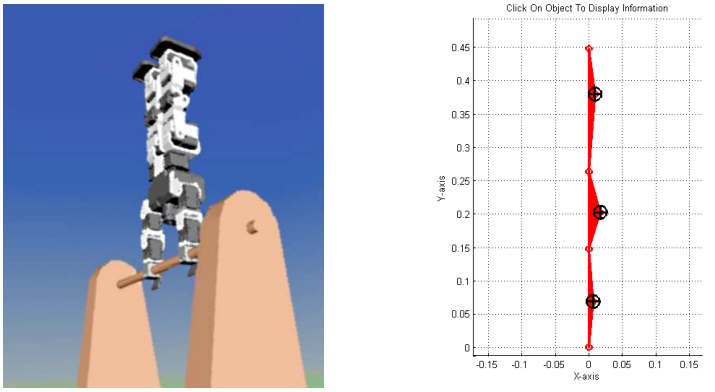


Figure 4.1: Drawings of the main blocks of the Gymnastic humanoid and its centers of gravity

position (y axis). Therefore the robot can only rotate along the z axis. Figure 4.2 represents the robot in the vertical unstable position along with its analog in the SimMechanics® simulator.



(a) Virtual Reality (b) Simulator representation

Figure 4.2: Humanoid in its vertical unstable position in Virtual Reality and in the SimMechanics simulator representation

Another important aspect to take in consideration are the limits of the angular displacements of the servos. The shoulders servos have a limit of 150° angular position range for each side. When the servo is in the middle position (150°, see Figure 2.9), it is assumed by the Simulator that the servo is at the zero degrees position. The hips servo however are not able to go back more than 40 encoders (11.7°) in order to prevent collisions. In the other direction it can go up to 150°.

	Mass (g)	Inertia Tensor (gcm^2)		
Arms	367.6	28479.93	59.0	−0.7
		59.0	21258.1	−1.3
		−0.7	−1.3	7890.7
Torso	981.5	37029.8	−1750.1	−54.7
		−1750.1	12211.3	134.6
		−54.7	134.6	32898.6
Legs	576.4	16420.4	1329.6	0.1
		1329.6	9032.2	1.4
		0.1	1.4	11328.0

Table 4.2: Mechanical properties of the main blocks of the Gymnastic humanoid

4.2 SimMechanics simulator

Using Simulink® and SimMechanics® it is possible to simulate the physical behavior of the humanoid. For our system it was used the SimMechanics® blocks of Matlab 2007a. Figure 4.3 shows the diagram block of the implemented simulator for the Gymnastic humanoid.

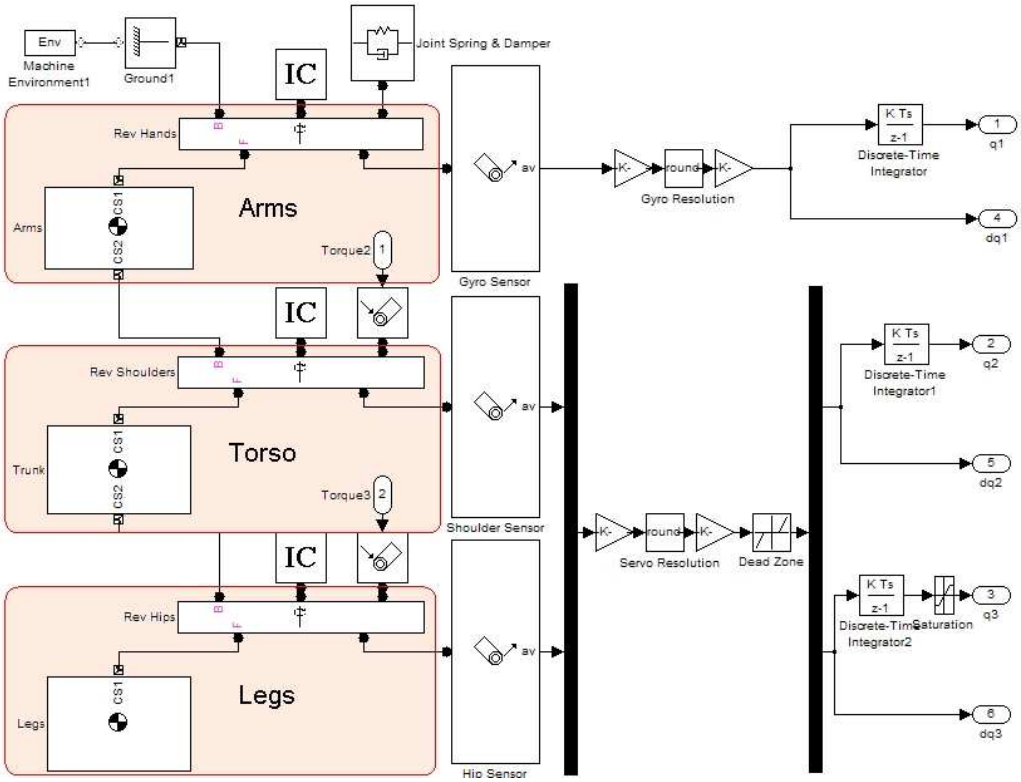


Figure 4.3: Gymnastic humanoid simulator plant

Friction and Gyroscope resolution

Using the gyroscope sensor it was possible to deduce not only the friction between the hands of the gymnast humanoid and the bar, but also the correspondence between the gyroscope output encoders and angular velocity in °/sec.

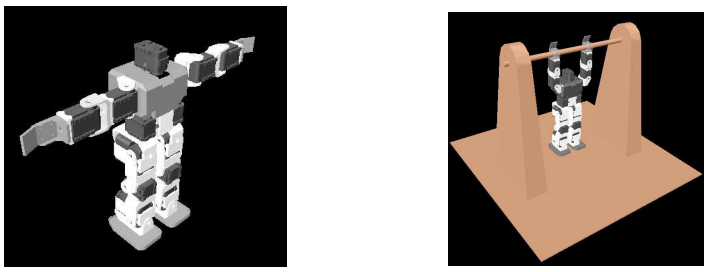
This experiment consisted by letting the humanoid fall from a knowing position and then observing the response of the gyroscope while the humanoid robot is swinging until it stops. Hereafter, friction models are used in the simulator in order to match the response of the gyroscope. Since the angular velocity given by the simulator is in °/sec, the correspondence to the gyroscope resolution is direct. The damper coefficient is about 0.3 and each step encoder of the gyroscopes is about 10°/sec.

Servos resolution

Servos have a 10 bit resolution for a full range of 300°, therefore each step of the servo encoder corresponds to 0.293°.

4.3 Virtual Reality animation

Virtual reality animation is a very important tool for simulation, since it gives a real perception of the behavior of a computer-simulated environment. Therefore and in order to have a smooth animation, a less detailed version of the Walking and Gymnastic humanoid models used in chapter 3 for identification, were exported to the virtual reality V-Realm Builder 2.0® program from its respective SolidWorks® 3D CAD models (Figure 4.4). To serve as background scenario, a thematic park was created as well (Figure 4.5). In this park, a humanoid can make use of its locomotion and stability control algorithms and of its artificial intelligent to do complex tasks such as resolving a maze, walking throughout a different types of floors and obstacles, running, climbing stairs, playing football, push objects, training its stability, skating or even doing an handstand on a high bar.

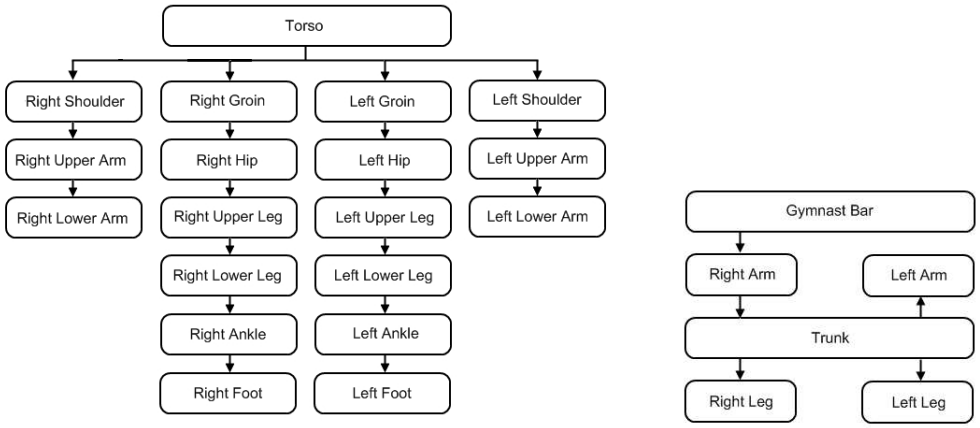


(a) Walking humanoid (b) Gymnastic humanoid

Figure 4.4: Virtual reality models of the Walking and Gymnastic humanoids

Figure 4.5: Humanoid virtual thematic park

The origin of each main block of the humanoid is coincident with its axis of rotation (joint). Therefore, by setting up in the virtual reality the real position of each block and the correspondent axis of rotation along with well defined parent-child hierarchy (Figure 4.6), the animation of the humanoid can be seen smooth and well defined. Table 4.3 and table 4.4 show in detail the exact position of each main block (or joint) and its axis of rotation, for both the Walking humanoid and the Gymnastic humanoid, respectively. Figure 3.2, shows the position and the axis of rotation of each joint.



(a) Walking humanoid

(b) Gymnastic humanoid

Figure 4.6: Parent-Child hierarchy for the Walking humanoid and the Gymnastic humanoid

Blocks	Rot axis	Pos x (mm)	Pos y (mm)	Posi z (mm)
Torso		0	0	0
Left Shoulder	z	0	0	–
Left Upper Arm	y	15	–	-76
Left Lower Arm	y	15	–	-114.25
Right Shoulder	z	0	0	–
Right Upper Arm	y	15	–	76
Right Lower Arm	y	15	–	114.25
Left Groin	y	-16	–	-33
Left Hip	x	–	-115.5	-33
Left Upper Leg	z	1	-115.5	–
Left Lower Leg	z	-14	-191	–
Left Ankle	z	1	-266.5	–
Left Foot	x	–	-266.5	-33
Right Groin	y	-16	–	33
Right Hip	x	–	-115.5	33
Right Upper Leg	z	1	-115.5	–
Right Lower Leg	z	-14	-191	–
Right Ankle	z	1	-266.5	–
Right Foot	x	–	-266.5	33

Table 4.3: Position of the main blocks and its orientation axis of the Walking humanoid

Blocks	Rot axis	Pos x (mm)	Pos y (mm)	Posi z (mm)
Gymnast Bar		6.5	144	0
Left Arm	z	6.5	144	–
Right Arm	z	6.5	144	–
Torso	z	0	0	–
Left Leg	z	1	-115.5	–
Right Leg	z	1	-115.5	–

Table 4.4: Position of the main blocks and its orientation axis of the Gymnastic humanoid

Chapter 5

Humanoid model

The Gymnastic humanoid robot can be seen as being compound of three main blocks. One block representing the arms, a second block representing the torso and a third block representing the legs. The joints of the robot are therefore the hands, the shoulders and the hips. Both shoulders and hips are actuated by two servos in each side. The hands are not actuated indeed, and therefore the system can be approximated by a three link underactuated pendulum (Figure 5.1).

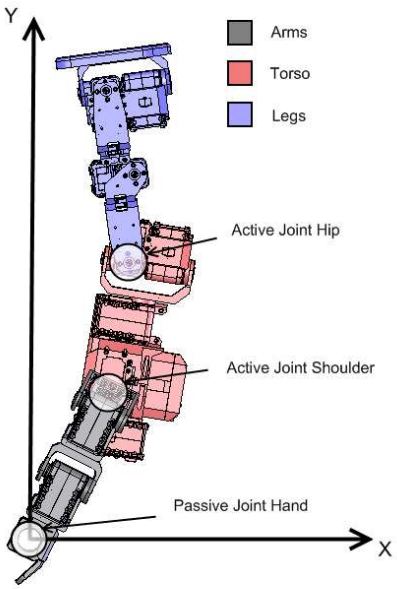


Figure 5.1: Active and Passive joints of the Gymnastic humanoid

In order to obtain the humanoid robot model the following steps will be taken:

1. Deduce the equations of motion of the system.
2. Linearize the system dynamics about the vertical unstable equilibrium and obtain state space representation model.
3. Analyze the system poles and zeros and confirm that it is a non-minimum phase system and therefore it is impossible to use classical cancelation techniques to stabilize it. For that reason an optimal control algorithm shall be adopted, namely the Linear Quadratic Regulator (LQR).
4. Use the emulation method with a zero order hold (ZOH), using different sampling times (To), and obtain the discrete-time model of the humanoid robot.

For simplification, it is assumed that the centers of gravity of all the blocks (arms, torso and legs) in the model of the humanoid, and therefore in its equations of motion, are aligned with the joints (e.g. case of ideal 3-link pendulum). In this way, it will be possible to test the robustness of the control strategy used.

5.1 Equations of Motion

In order to describe the dynamic behavior of this multi-body robotic system, it is necessary to determine its equations of motion. These equations can be derived from the classical Euler-Lagrange equations, according to Figure 5.2.

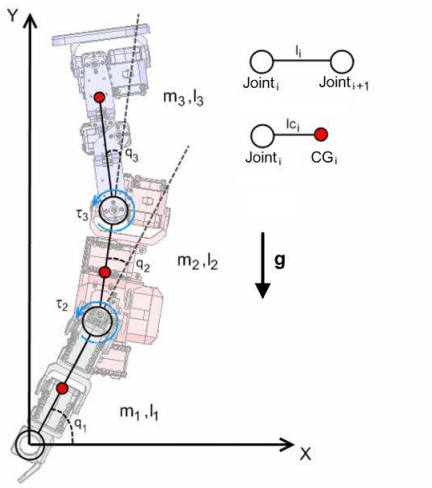


Figure 5.2: Representation of the humanoid seen as an underactuated triple pendulum.

Where:

- q_i is the angle of joint i in respect to the previous link.
- m_i is the mass of block i.
- I_i is the inertia of block i.
- τ_i is the torque actuated on the active joint i.
- l_i is the length between joint i and joint i+1.
- lc_i is the length between joint i and the center of gravity of the mass i.

Deduction of Euler-Lagrange equations:

$$\begin{aligned} x_1 &= lc_1 \cos(q_1) \\ y_1 &= lc_1 \sin(q_1) \end{aligned} \tag{5.1}$$

$$\begin{aligned} x_2 &= l_1 \cos(q_1) + lc_2 \cos(q_1 + q_2) \\ y_2 &= l_1 \sin(q_1) + lc_2 \sin(q_1 + q_2) \end{aligned} \tag{5.2}$$

$$\begin{aligned} x_3 &= l_1 \cos(q_1) + l_2 \cos(q_1 + q_2) + lc_3 \cos(q_1 + q_2 + q_3) \\ y_3 &= l_1 \sin(q_1) + l_2 \sin(q_1 + q_2) + lc_3 \sin(q_1 + q_2 + q_3) \end{aligned} \tag{5.3}$$

Where:

- x_i and y_i are the coordinates of the center of gravity of mass i.

For a n-link pendulum:

$$\begin{aligned} x_i &= \sum_{j=1}^{i-1} \left[l_j \cos\left(\sum_{k=1}^{j} q_k \right) \right] + lc_i \cos\left(\sum_{k=1}^{i} q_k \right), \quad i = 1, 2, ..., n \\ y_i &= \sum_{j=1}^{i-1} \left[l_j \sin\left(\sum_{k=1}^{j} q_k \right) \right] + lc_i \sin\left(\sum_{k=1}^{i} q_k \right), \quad i = 1, 2, ..., n \end{aligned} \tag{5.4}$$

The Kinetic energy (T) of each element (mass) of the system:

$$\begin{aligned} T_1 &= \frac{1}{2} \left[m_1 \left(\dot{x}_1^2 + \dot{y}_1^2 \right) + I_1 \dot{q}_1^2 \right] \\ T_2 &= \frac{1}{2} \left[m_2 \left(\dot{x}_2^2 + \dot{y}_2^2 \right) + I_2 \left(\dot{q}_1 + \dot{q}_2 \right)^2 \right] \\ T_3 &= \frac{1}{2} \left[m_3 \left(\dot{x}_3^2 + \dot{y}_3^2 \right) + I_3 \left(\dot{q}_1 + \dot{q}_2 + \dot{q}_3 \right)^2 \right] \end{aligned} \tag{5.5}$$

The Potential energy (U) of each element (mass) of the system:

$$\begin{aligned} U_1 &= m_1 g y_1 \\ U_2 &= m_2 g y_2 \\ U_3 &= m_3 g y_3 \end{aligned} \tag{5.6}$$

The Lagrangian of the system is therefore given by:

$$L = \sum_{i=1}^{3} (T_i - U_i) \tag{5.7}$$

For a n-link pendulum:

$$\begin{gathered} T_i = \frac{1}{2}\left[m_i \left(\dot{x}_i^2 + \dot{y}_i^2 \right) + I_i \left(\sum_{k=1}^{i} \dot{q}_k \right)^2 \right], \quad i = 1, 2, ..., n \\ U_i = m_i g y_i, \quad i = 1, 2, ..., n \\ L = \sum_{i=1}^{n} (T_i - U_i) \end{gathered} \tag{5.8}$$

Solving the Lagrange equations for each element of the system, the following equations of motion result:

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} + \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} + \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_2 \\ \tau_3 \end{bmatrix} \tag{5.9}$$

Where:

$$
\begin{aligned}
m_{11} &= m_1 lc_1^2 + m_2 l_1^2 + m_2 lc_2^2 + m_3 l_1^2 + m_3 l_2^2 + m_3 lc_3^2 \\
&\quad + 2m_2 l_1 lc_2 \cos(q_2) + 2m_3 l_1 l_2 \cos(q_2) + 2m_3 l_2 lc_3 \cos(q_3) \\
&\quad + 2m_3 l_1 lc_3 \cos(q_2 + q_3) + I_1 + I_2 + I_3
\end{aligned} \tag{5.10a}
$$

$$
\begin{aligned}
m_{12} &= m_2 lc_2^2 + m_3 l_2^2 + m_3 lc_3^2 + m_2 l_1 lc_2 \cos(q_2) + m_3 l_1 l_2 \cos(q_2) \\
&\quad + 2m_3 l_2 lc_3 \cos(q_3) + m_3 l_1 lc_3 \cos(q_2 + q_3) + I_2 + I_3
\end{aligned} \tag{5.10b}
$$

$$m_{13} = m_3 lc_3^2 + m_3 l_2 lc_3 \cos(q_3) + m_3 l_1 lc_3 \cos(q_2 + q_3) + I_3 \tag{5.10c}$$

$$m_{21} = m_{12} \tag{5.10d}$$

$$m_{22} = m_2 lc_2^2 + m_3 l_2^2 + m_3 lc_3^2 + 2m_3 l_2 lc_3 \cos(q_3) + I_2 + I_3 \tag{5.10e}$$

$$m_{23} = m_3 lc_3^2 + m_3 l_2 lc_3 \cos(q_3) + I_3 \tag{5.10f}$$

$$m_{31} = m_{13} \tag{5.10g}$$

$$m_{32} = m_{23} \tag{5.10h}$$

$$m_{33} = m_3 lc_3^2 + m_3 l_2 lc_3 \cos(q_3) + I_3 \tag{5.10i}$$

$$
\begin{aligned}
\phi_1 &= (m_1 lc_1 + m_2 l_1 + m_3 l_1) g \cos(q_1) + (m_2 lc_2 + m_3 l_2) g \cos(q_1 + q_2) \\
&\quad + m_3 lc_3 g \cos(q_1 + q_2 + q_3)
\end{aligned} \tag{5.11a}
$$

$$\phi_2 = (m_2 lc_2 + m_3 l_2) g \cos(q_1 + q_2) + m_3 lc_3 g \cos(q_1 + q_2 + q_3) \tag{5.11b}$$

$$\phi_3 = m_3 lc_3 g \cos(q_1 + q_2 + q_3) \tag{5.11c}$$

and,

$$
\begin{aligned}
h_1 &= -m_2 l_1 lc_2 \dot{q}_2 (2\dot{q}_1 + \dot{q}_2) \sin(q_2) \\
&\quad - m_3 l_1 l_2 \dot{q}_2 (2\dot{q}_1 + \dot{q}_2) \sin(q_2) \\
&\quad - m_3 l_2 lc_3 \dot{q}_3 (2\dot{q}_1 + 2\dot{q}_2 + \dot{q}_3) \sin(q_3) \\
&\quad - m_3 l_1 lc_3 (\dot{q}_2 + \dot{q}_3)(2\dot{q}_1 + \dot{q}_2 + \dot{q}_3) \sin(q_2 + q_3)
\end{aligned} \tag{5.12a}
$$

$$
\begin{aligned}
h_2 &= m_2 l_1 lc_2 \dot{q}_1^2 \sin(q_2) + m_3 l_1 l_2 \dot{q}_1^2 \sin(q_2) \\
&\quad + m_3 l_1 lc_3 \dot{q}_1^2 \sin(q_2 + q_3) \\
&\quad - m_3 l_2 lc_3 \dot{q}_3 (2\dot{q}_1 + 2\dot{q}_2 + \dot{q}_3) \sin(q_3)
\end{aligned} \tag{5.12b}
$$

$$
\begin{aligned}
h_3 &= m_3 l_1 lc_3 \dot{q}_1^2 \sin(q_2 + q_3) \\
&\quad + m_3 l_2 lc_3 (\dot{q}_1 + \dot{q}_2)^2 \sin(q_3)
\end{aligned} \tag{5.12c}
$$

Where:

- m_{ij} are the inertial terms.
- ϕ_i are the gravitational terms.

- h_i are the Corriolis and the centrifugal terms.
- τ_i are the input torques.

For a n-link pendulum the equations of motion would be:

$$\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \vdots \\ \ddot{q}_n \end{bmatrix} + \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{bmatrix} + \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_n \end{bmatrix} \tag{5.13}$$

Where:

$$m_{ij} = \sum_{k=j}^{n} \left\{ m_k \left[\sum_{a=j}^{k} (l_a) + \sum_{b=i}^{k} \sum_{c=j \wedge j \neq i}^{k} l_b l_c \cos \left(\sum_{d=1}^{|b-c|} q_{d+\min(b,c)} \right) \right] + I_k \right\}, \quad i \leq j$$
$$m_{ij} = m_{ji}, \quad i > j \tag{5.14}$$

$$\phi_i = g \sum_{k=i}^{n} \sum_{a=1}^{k} m_k l_a \cos \left(\sum_{b=1}^{a} q_b \right) \tag{5.15}$$

$$h_i = C_i + \sum_{j=i}^{n} \dot{m}_{ij}$$

with

$$C_i = \sum_{k=i}^{n} \sum_{b=i}^{k} \sum_{c=1 \wedge c \neq i}^{k} m_k l_b l_c \left(\sum_{d=1}^{\min(b,c)} \dot{q}_d \right)^2 \sin \left(\sum_{d=1}^{|b-c|} q_{d+\min(b,c)} \right) \text{iff } \min(b,c) < 1$$
$$\dot{m}_{ij} = -\sum_{k=j}^{n} \sum_{b=i}^{k} \sum_{c=j \wedge j \neq i}^{k} m_k l_b l_c \left(\sum_{d=1}^{|b-c|} \dot{q}_{d+\min(b,c)} \right) \sin \left(\sum_{d=1}^{|b-c|} q_{d+\min(b,c)} \right) \text{iff } \min(b,c) \geq 1 \tag{5.16}$$

and,

$$l_i = \begin{cases} lc_i & i = k \\ l_i & i \neq k \end{cases} \tag{5.17}$$

5.2 Linearization

The above equations of motion of the system are highly nonlinear. Nevertheless, the goal of the controller is to stabilize the humanoid at the vertical unstable equilibrium.

$$q = \left[\frac{\pi}{2}, 0, 0 \right]^T$$

$$\dot{q} = [0, 0, 0]^T$$

Hence, for this situation in particular, the system can be linearized and controlled by linear control algorithms.

Using a first order Taylor's expansion of (5.9) results the following linearized system for the vertical unstable equilibrium.

$$M\ddot{q} - \Phi q = T\tau$$

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} - \begin{bmatrix} \phi_{11} & \phi_{12} & \phi_{13} \\ \phi_{21} & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} \begin{bmatrix} q_1 - \frac{\pi}{2} \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tau_2 \\ \tau_3 \end{bmatrix} \tag{5.18}$$

Note that the h_i terms disappeared in the linearization.

Expanding each element of (5.18) results:

$$\begin{aligned} m_{11} = {} & m_1 lc_1^2 + m_2 l_1^2 + m_2 lc_2^2 + m_3 l_1^2 + m_3 l_2^2 + m_3 lc_3^2 \\ & + 2m_2 l_1 lc_2 + 2m_3 l_1 l_2 + 2m_3 l_2 lc_3 \\ & + 2m_3 l_1 lc_3 + I_1 + I_2 + I_3 \end{aligned} \tag{5.19a}$$

$$\begin{aligned} m_{12} = {} & m_2 lc_2^2 + m_3 l_2^2 + m_3 lc_3^2 + m_2 l_1 lc_2 + m_3 l_1 l_2 \\ & + 2m_3 l_2 lc_3 + m_3 l_1 lc_3 + I_2 + I_3 \end{aligned} \tag{5.19b}$$

$$m_{13} = m_3 lc_3^2 + m_3 l_2 lc_3 + m_3 l_1 lc_3 + I_3 \tag{5.19c}$$

$$m_{21} = m_{12} \tag{5.19d}$$

$$m_{22} = m_2 lc_2^2 + m_3 l_2^2 + m_3 lc_3^2 + 2m_3 l_2 lc_3 + I_2 + I_3 \tag{5.19e}$$

$$m_{23} = m_3 lc_3^2 + m_3 l_2 lc_3 + I_3 \tag{5.19f}$$

$$m_{31} = m_{13} \tag{5.19g}$$

$$m_{32} = m_{23} \tag{5.19h}$$

$$m_{33} = m_3 lc_3^2 + m_3 l_2 lc_3 + I_3 \tag{5.19i}$$

$$\phi_{11} = (m_1 lc_1 + m_2 l_1 + m_3 l_1 + m_2 lc_2 + m_3 l_2 + m_3 lc_3)g \tag{5.20a}$$

$$\phi_{12} = (m_2 lc_2 + m_3 l_2 + m_3 lc_3)g \tag{5.20b}$$

$$\phi_{13} = (m_3 lc_3)g \tag{5.20c}$$

$$\phi_{22} = \phi_{21} = \phi_{12} \tag{5.20d}$$

$$\phi_{33} = \phi_{31} = \phi_{32} = \phi_{23} = \phi_{13} \tag{5.20e}$$

For a n-link pendulum:

$$M\ddot{q} - \Phi q = T\tau$$

$$\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ m_{21} & m_{22} & \cdots & m_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nn} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} - \begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} \end{bmatrix} \begin{bmatrix} q_1 - \frac{\pi}{2} \\ q_2 \\ \vdots \\ q_n \end{bmatrix} = T_{n\times n} \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_n \end{bmatrix} \tag{5.21}$$

Note: The columns in the matrix T that corresponds to passive joints should be eliminated (see equation (5.18)).

Expanding 5.21 results:

$$m_{ij} = \sum_{k=j}^{n} \left\{ m_k \left[\sum_{a=j}^{k} (l_a) + \sum_{b=i}^{k} \sum_{c=j \wedge j \neq i}^{k} l_b l_c \right] + I_k \right\}, \quad i \leq j$$
$$m_{ij} = m_{ji}, \quad i > j \tag{5.22}$$

$$\phi_{ii} = g \sum_{k=i}^{n} \sum_{a=1}^{k} m_k l_a, \quad i = 1, 2, ..., n$$
$$\phi_{ij} = \phi_{ji} = \phi_{jj}, \quad i \leq j \tag{5.23}$$

and

$$l_i = \begin{cases} lc_i & i = k \\ l_i & i \neq k \end{cases} \tag{5.24}$$

Note:

$$\begin{aligned} \cos(q_1 + Q) &= \cos(q_1)\cos(Q) - \sin(q_1)\sin(Q) \\ &= -\sin\left(q_1 - \tfrac{\pi}{2}\right)\cos(Q) - \sin(q_1)\sin(Q) \\ &\cong -\left(q_1 - \tfrac{\pi}{2}\right) - Q \\ when: \quad & q_1 \to \tfrac{\pi}{2}; \quad Q \to 0 \end{aligned}$$

5.3 Continuous state space model

Given the above linearization of a generalized n-link pendulum system, the state space model is easily obtained.

Let $x = \begin{bmatrix} q_1 - \frac{\pi}{2} & q_2 & \cdots & q_n & \dot{q}_1 & \dot{q}_2 & \cdots & \dot{q}_n \end{bmatrix}^T$ be the state vector and u $= \begin{bmatrix} \tau_1 & \tau_2 & \cdots & \tau_n \end{bmatrix}^T$ the input vector. Let m be the order of the system and l the number of columns of matrix T. Then the matrices A,B,C,D of the state space model representation:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t) \end{aligned} \tag{5.25}$$

are giving respectively by:

$$\begin{aligned} A &= \begin{bmatrix} 0_{n\times n} & I_{n\times n} \\ M^{-1}\Phi & 0_{n\times n} \end{bmatrix} \\ B &= \begin{bmatrix} 0_{n\times l} \\ M^{-1}T \end{bmatrix} \\ C &= I_{m\times m} \\ D &= 0_{m\times l} \end{aligned} \tag{5.26}$$

From the mechanical properties we have the following parameters:

	$l\ (mm)$	$lc\ (mm)$	$m\ (g)$	$I\ (gcm^2)$
Link 1 (Arms)	143.6	68.7	367.6	7890.7
Link 2 (Torso)	115.8	57.5	981.5	32898.6
Link 3 (Legs)	184.0	116.3	576.4	11328.0

Table 5.1: Physical properties of the gymnast humanoid

Substituting these values in the linearized equations of motion (equations (5.19) and (5.20)), we obtain:

$$
A = \begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
74.2389 & -117.8764 & 0.0028 & 0 & 0 & 0 \\
-81.5857 & 348.7777 & -76.0776 & 0 & 0 & 0 \\
7.3471 & -230.9113 & 215.8591 & 0 & 0 & 0
\end{bmatrix}
$$

$$
B = \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
-176.2065 & 97.5517 \\
527.7831 & -467.2635 \\
-467.2635 & 697.9566
\end{bmatrix} \tag{5.27}
$$

$$
C = I_{6\times 6}
$$

$$
D = 0_{6\times 2}
$$

The eigenvalues of the matrix A are the poles of the system. Thus, the poles are:

$$
Poles = \begin{bmatrix} -21.2157 & -12.5086 & -5.6837 & 5.6837 & 12.5086 & 21.2157 \end{bmatrix} \tag{5.28}
$$

The zeros can be determined by finding the transfer function between each output and input. The full list can be found in Appendix E.

From the analysis of the position of the poles and zeros of the system model, it can be seen that there are zeros and poles in the right half plane, and thus the system is unstable and presents a non-minimum phase behavior. Classical cancelation techniques are then inefficient to stabilize the system. Taking this in consideration, an optimal control strategy will be adopted to control the system.

5.4 Discrete state space model

In order to control the humanoid robot through the PC, it is necessary to design a digital controller. This can be easily done by using the emulation method and assuming a zero order hold (ZOH) for the input to the continuous system.

The discrete state-space model takes the following form:

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k) \\ y(k) &= C_d x(k) + D_d u(k) \end{aligned} \tag{5.29}$$

Where the discrete matrices A_d and B_d can be deduced from the continues state-space model (5.26) by integrating the state evolution through time:

$$x(t) = e^{A(t-t_0)} x(t_0) + \int_{t_0}^{t} e^{A(t-\tau)} B u(\tau)\, d\tau \tag{5.30}$$

Assuming T_s as the sample time of the discrete system, $t_0 = kT_s$ and $t = t_0 + T_s$, then equation (5.30) takes the form:

$$x((k+1)T_s) = e^{AT_s} x(kT_s) + \int_{kT_s}^{(k+1)T_s} e^{A[(k+1)T_s - \tau]} B u(\tau)\, d\tau \tag{5.31}$$

Assuming now that the input of the system is constant over the integration interval:

$$x((k+1)T_s) = e^{AT_s} x(kT_s) + \left(\int_0^{T_s} e^{A\tau} d\tau \right) B u(kT_s) \tag{5.32}$$

Finally results:

$$\begin{cases} A_d = e^{AT_s} \\ B_d = \left(\int_0^{T_s} e^{A\tau} d\tau \right) B \end{cases} \tag{5.33}$$

(c.f. chapter 5.1.2 in (Ayala Botto, 2003b))

Matrices C and D do not change when converted to the discrete time domain. Hence $C_d = C$ and $D_d = D$.

5.4.1 Sample time determination

By analyzing the position of the dominant poles of the system from Figure E.1 and Figure E.2, the main natural frequency of the system can be attained:

$$w_n = 5.6592 \text{ rad/s} \tag{5.34}$$

And then, the biggest component of the frequency is:

$$f = 5.6592/2\pi = 0.90 \text{ Hz} \tag{5.35}$$

This means that the frequency of 100 Hz imposed seems a reasonable one since it is approximately 2000 times biger. Hence, we are going to use a sample time of 0.01 sec and also another of 0.02 sec (50 Hz) for comparison.

As a result, the matrices A_d and B_d from the discrete-time state-space model for T_s=0.01 sec and T_s=0.02 sec are given by expressions (5.36) and (5.37), respectively:

$$A_d = \begin{bmatrix} 1.0037 & -0.0059 & 0 & 0.01 & 0 & 0 \\ -0.0041 & 1.0175 & -0.0038 & 0 & 0.0101 & 0 \\ 0.0004 & -0.0116 & 1.0108 & 0 & 0 & 0.01 \\ 0.7449 & -1.1871 & 0.0015 & 1.0037 & -0.0059 & 0 \\ -0.8217 & 3.5252 & -0.7623 & -0.0041 & 1.0175 & -0.0038 \\ 0.0770 & -2.3535 & 2.1587 & 0.0004 & -0.0116 & 1.0108 \end{bmatrix}$$

$$B_d = \begin{bmatrix} -0.0088 & 0.0049 \\ 0.0265 & -0.0235 \\ -0.0235 & 0.035 \\ -1.7746 & 0.9859 \\ 5.3169 & -4.7101 \\ -4.7101 & 7.0229 \end{bmatrix} \tag{5.36}$$

$$A_d = \begin{bmatrix} 1.0149 & -0.0239 & 0.0001 & 0.0201 & -0.0002 & 0 \\ -0.0166 & 1.0708 & -0.0155 & -0.0001 & 0.0205 & -0.0001 \\ 0.0016 & -0.0471 & 1.0436 & 0 & -0.0003 & 0.0203 \\ 1.5051 & -2.4246 & 0.0122 & 1.0149 & -0.0239 & 0.0001 \\ -1.6789 & 7.1758 & -1.5794 & -0.0166 & 1.0708 & -0.0155 \\ 0.1752 & -4.7949 & 4.4034 & 0.0016 & -0.0471 & 1.0436 \end{bmatrix}$$

$$B_d = \begin{bmatrix} -0.0357 & 0.0199 \\ 0.1071 & -0.095 \\ -0.095 & 0.1413 \\ -3.6254 & 2.035 \\ 10.8705 & -9.6467 \\ -9.6467 & 14.3077 \end{bmatrix} \tag{5.37}$$

5.4.2 Reachability and observability

The system is completely reachable and observable, if the rank of the reachability and observability matrices is to the order of the system.

Reachability matrix:

$$\mathcal{C} = \begin{bmatrix} B_d & A_d B_d & A_d^2 B_d & \cdots & A_d^{(m-1)} B_d \end{bmatrix} \tag{5.38}$$

Observability matrix:

$$\mathcal{O} = \begin{bmatrix} C_d \\ C_d A_d \\ C_d A_d^2 \\ \vdots \\ C_d A_d^{(m-1)} \end{bmatrix} \tag{5.39}$$

In our problem, the rank of each matrix is 6 for both configurations, (5.36) and (5.37), which means that the system is reachable and observable.

Chapter 6

Humanoid Control and Simulation Results

In this chapter, a control situation is presented. The objective is to have the humanoid robot doing a handstand on a high bar, i.e. behaving like an underactuated triple pendulum.

The control of the humanoid in a high bar can be divided into two individual phases. The first phase is the swing-up control, where the robot is ought from the down vertical stable position to the up vertical unstable position. The other phase is to control the humanoid in the unstable vertical position (balancing control).

The Linear Quadratic Regulator (LQR) will be adopted throughout (Lancaster and Rodman, 1995). Linear Quadratic Regulator is a feedback controller for solving dynamic systems of linear differential equations by minimizing a quadratic function cost. This cost can be seen as the sum of undesired deviations from the optimal value.

6.1 Linear Quadratic Regulator (LQR)

This control technique provides a linear state feedback control law for the system. This law has the following form:

$$u = -k^T x \tag{6.1}$$

where k is a $m \times l$ matrix that contains the state feedback gains (Kalman gains) and can be obtained by minimizing the following performance index:

$$J = \int_0^\infty \left(x^T Q x + u^T R u\right) dt \tag{6.2}$$

Where Q and R are the "weighting matrices". Q is a $m \times m$ matrix and R is a $l \times l$ one.

The solution of equation (6.2) is found by solving to P (Lyapunov function matrix) the Algebraic Riccati Equation (6.3) and then equation (6.4). A and B are the A and B matrices of the humanoid state-space model.

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{6.3}$$

$$k^T = R^{-1}B^T P \tag{6.4}$$

For our system we use Q as the 6×6 identity matrix and R as the 2×2 identity matrix. Therefore we have for the model (5.27):

$$k^T = \begin{bmatrix} -146.7206 & -63.3382 & -23.2078 & -25.2863 & -11.8534 & -5.0644 \\ -92.6369 & -41.0629 & -13.3245 & -15.8651 & -8.1298 & -2.2096 \end{bmatrix} \tag{6.5}$$

6.2 LQR Simulation Results

The plots shown in Figure 6.2 (a,c,e) show the behavior of the LQR controlled system when the state vector is defined as $x = \begin{bmatrix} q_1 - \frac{\pi}{2} & q_2 & q_3 & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 \end{bmatrix}^T$. As it can be seen, the system could not be stabilized. The reason is due to the fact that the real position of the centers of gravity of the arms, torso and legs were not taken into account when building the model (c.f. chapter 5.1). Nevertheless, a simple solution for this problem consists in compensating the system with a angle α obtained from the resultant center of gravity of the three bodies about the vertical (Figure 6.1). the new state vector is then $x = \begin{bmatrix} q_1 - (\frac{\pi}{2} + \alpha) & q_2 & q_3 & \dot{q}_1 & \dot{q}_2 & \dot{q}_3 \end{bmatrix}^T$. The controller is now able to stabilize the system (Figure 6.2 (b,d,f)).

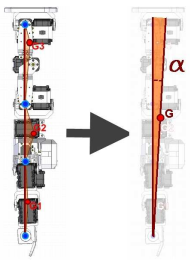
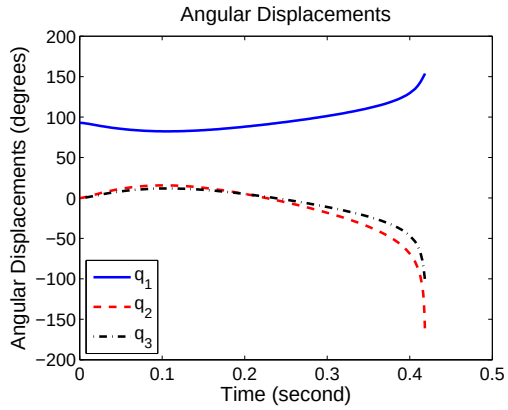
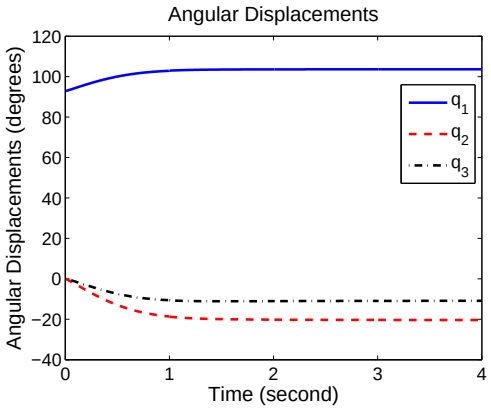


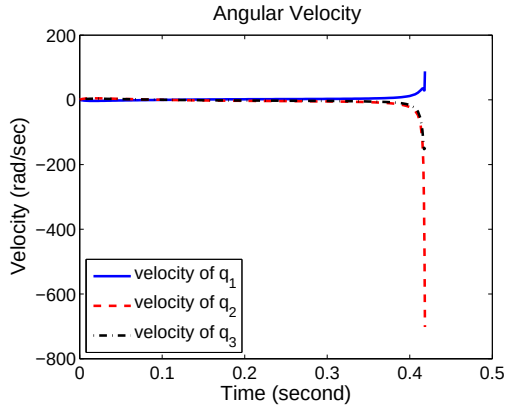
Figure 6.1: Position of the Centers of gravity of the arms, torso and legs and its equivalent one



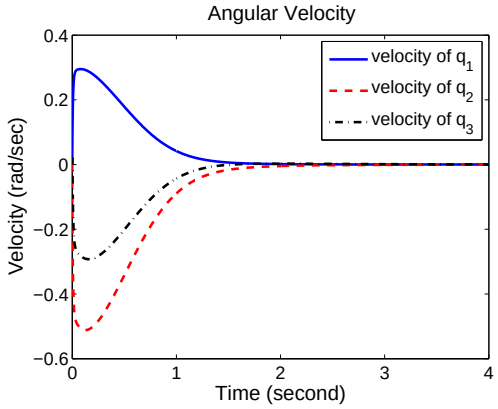
(a) (A) Without angle compensation



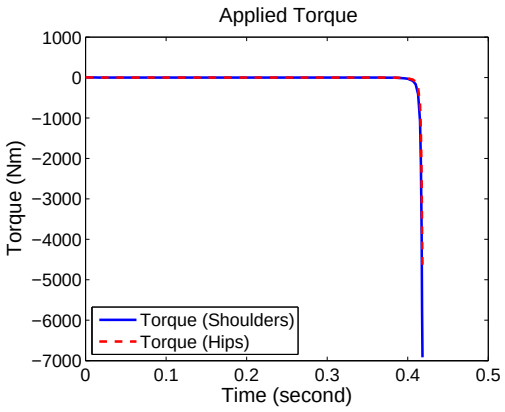
(b) (B) With angle compensation



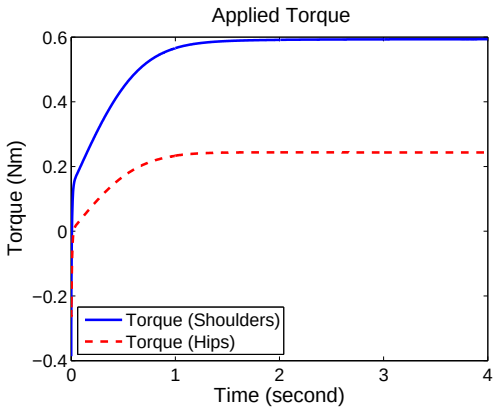
(c) (A) Without angle compensation



(d) (B) With angle compensation



(e) (A) Without angle compensation



(f) (B) With angle compensation

Figure 6.2: Simulation using Linear Quadratic Regulator for balancing without angle compensation (A) and with (B)

6.3 Discrete Linear Quadratic Regulator (DLQR)

The implementation of the DLQR controller (Figure 6.3) requires a discrete-time model of the system to be controlled. Its formulation is practically the same as for the continuous-time case. It has the same objective of finding the optimal feedback gains or Kalman gains, which optimize a discrete time version of the performance index given in (6.2). The solution will be based on the discrete-time Riccati Equation (6.6), plus finding the optimal gains given by (6.7).

$$A_d^T[P - PB_d(B_d^T PB_d + R)^{-1}B_d^T P]A_d + Q = P \tag{6.6}$$

$$k^T = (B_d^T PB_d + R)^{-1}B_d^T P \tag{6.7}$$

Next, it is presented the discrete optimal feedback gains when T_s=0.01 sec and T_s=0.02 sec (equations (6.8) and (6.9) respectively) for the model (5.36) and (5.37) respectively, with Q being the identity matrix times 100 and R being the identity matrix.

$$k_d^T\big|_{T=0.01s} = \begin{bmatrix} -75.7584 & -32.1872 & -11.4543 & -13.0048 & -6.2134 & -2.3442 \\ -62.057 & 26.9283 & -9.0351 & -10.6539 & -5.1619 & -1.8225 \end{bmatrix} \tag{6.8}$$

$$k_d^T\big|_{T=0.02s} = \begin{bmatrix} -37.8516 & -15.5385 & -5.5276 & -6.4685 & -3.0698 & -1.154 \\ -31.0699 & -13.2839 & -4.2395 & -5.3157 & -2.5617 & -0.8983 \end{bmatrix} \tag{6.9}$$

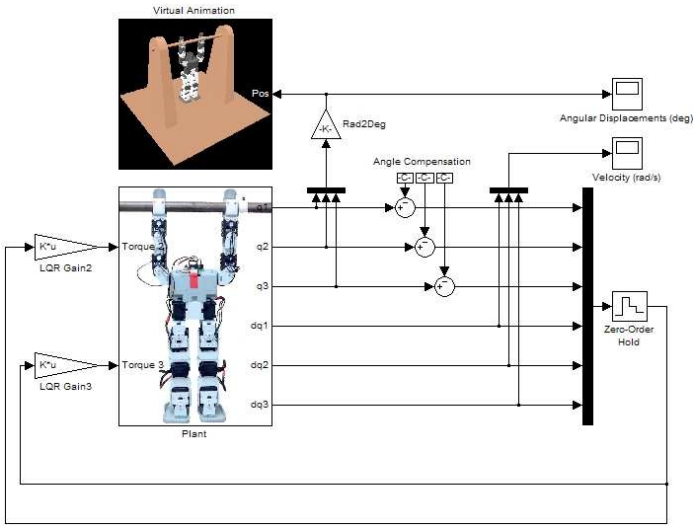


Figure 6.3: Implemented DLQR controller

6.4 DLQR Simulation Results

From Figure 6.4, it is noticeable that it is indeed possible to control the humanoid such that it is able to do the hand-stand on a high bar. Recall the fact that the maximum allowed torque of the servos are 1.0 Nm. Since each joint has two servos, then the maximum torque is 2.0 Nm, which is never reached during the simulation. Also the angular displacements are inside the allowable intervals.

Adopting the sampling time of 0.01 seconds, different simulations were made. In the plots of Figure 6.4 it was used only the angle compensation in the first joint. Experiments show, however, that by introducing angles compensation for the other joints too, the final torque input would be smaller and the humanoid final configuration more appealing (stretcher). For these experiments, the best angles compensation are shown in table 6.1.

Joint	Angle Compensation (rad)
Hands	0.04
Shoulders	0.05
Hips	-0.05

Table 6.1: Best Angle Compensation

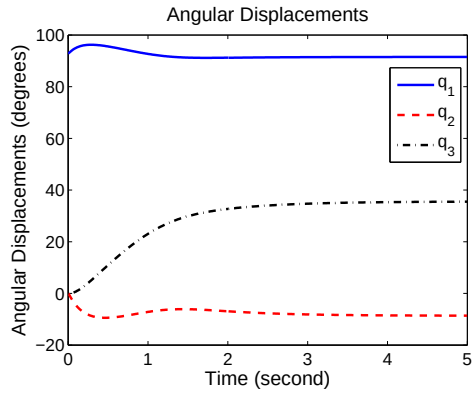
The plots in Figure 6.5(a,c,e) show the evolution of the system when these angles compensation are applied. As it can be seen the input torques are smaller, being the system more stable. In the rest of the simulations, it is used the same angles compensations.

The plots in Figure 6.5(b,d,f) show the behavior of the system when the angular position resolution of a real servo (10 bits resolution for 300 degrees) is used. As it can be seen, the system starts to present some chattering, although the controller is able to stabilize it.

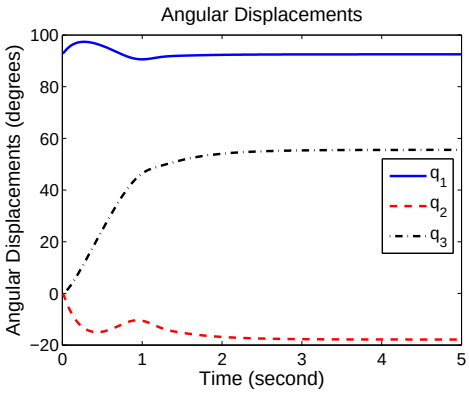
The controller starts to present some difficulty in terms of stabilizing the system when the gyroscope resolution is added to the simulator. Since the resolution of the gyro is only $10^o/s$, the system presents a higher amplitude input control actions in order to compensate the lack of information from the gyro. Nevertheless, the controller is still able to stabilize the system, even with constant disturbances (plots in Figure 6.6(a,c,e)).

If the dead zone of the servos for lower velocities (10% of the full range of 10 bits resolution for 300^o) are considered, then it can be seen from the plots in Figure 6.6(b,d,f) that the controller is not able to stabilize the system. Moreover, the system input torques would saturate since the maximum allowable value is 2 Nm.

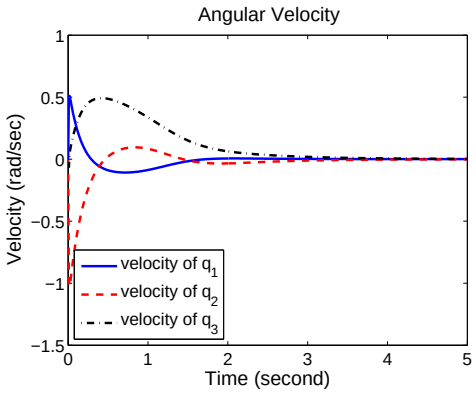
By adding the friction effect of the hands on the bar, the system destabilizes completely in less then 6 seconds (plots of the Figure 6.7).



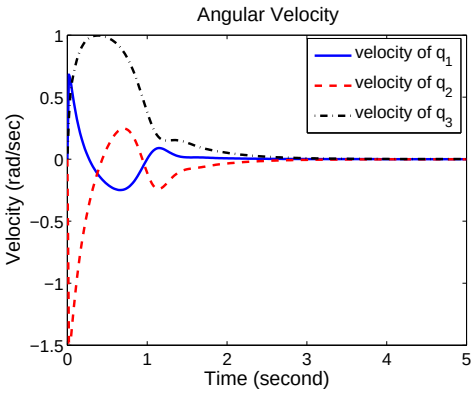
(a) (A) T_s=0.01 sec



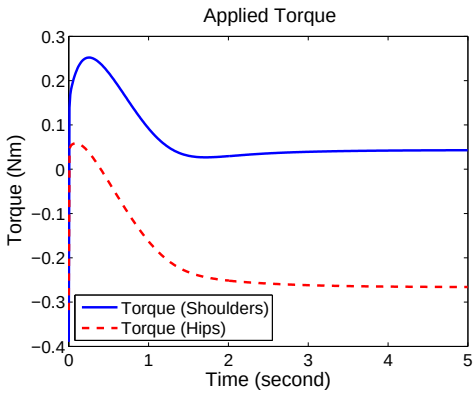
(b) (B) T_s=0.02 sec



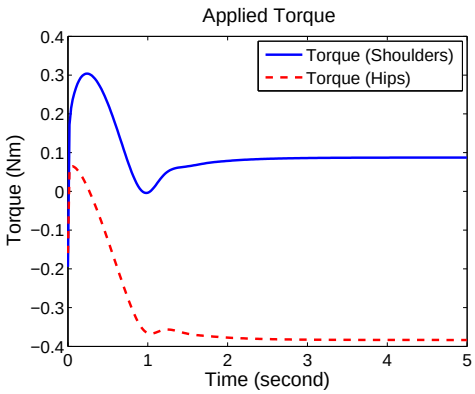
(c) (A) T_s=0.01 sec



(d) (B) T_s=0.02 sec

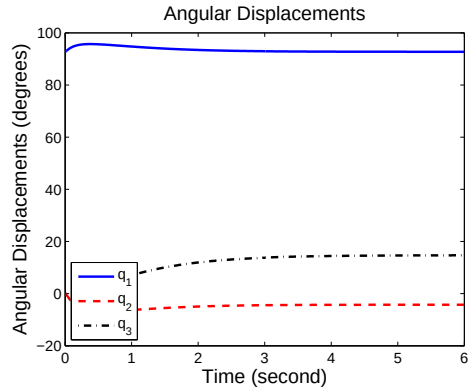


(e) (A) T_s=0.01 sec

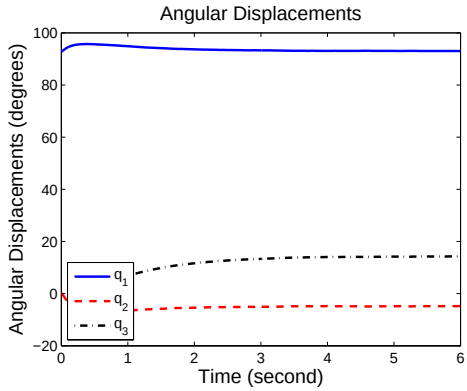


(f) (B) T_s=0.02 sec

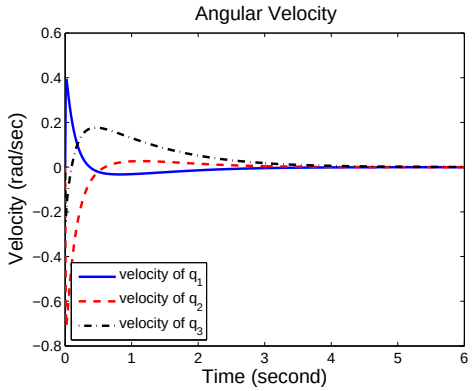
Figure 6.4: Simulation using discrete Linear Quadratic Regulator for balancing with T_s=0.01 sec (A) and T_s=0.02 sec (B)



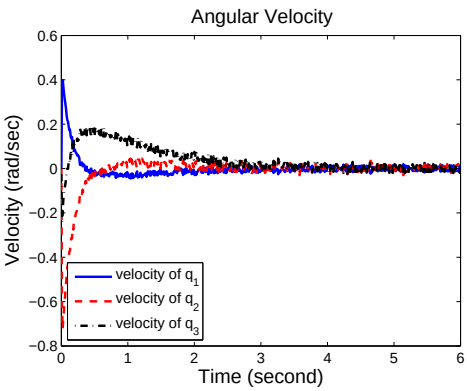
(a) (A) T_s=0.01 sec



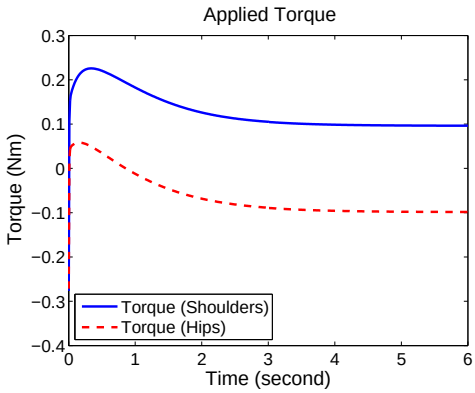
(b) (B) T_s=0.01 sec



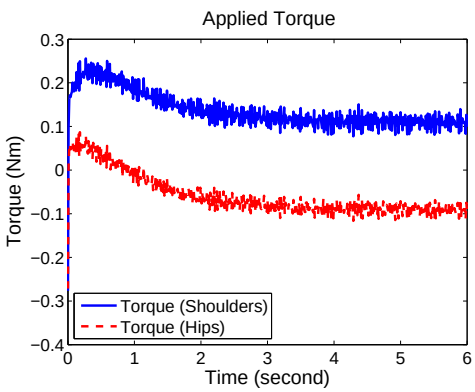
(c) (A) T_s=0.01 sec



(d) (B) T_s=0.01 sec

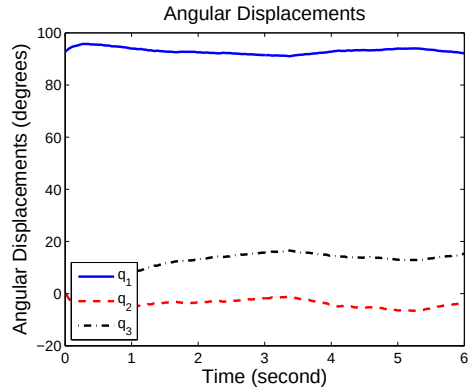


(e) (A) T_s=0.01 sec

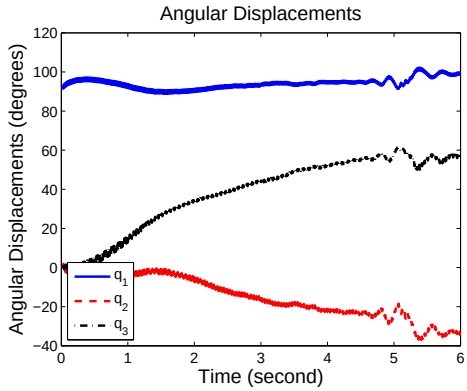


(f) (B) T_s=0.01 sec

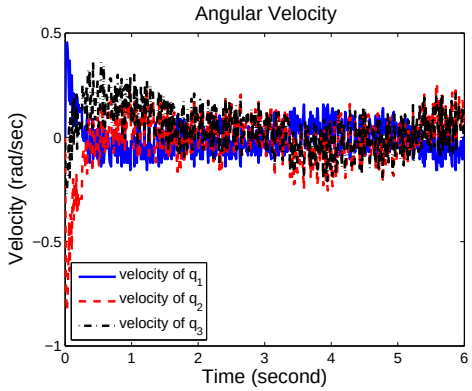
Figure 6.5: Simulation using angles compensation for the three joint (A) and simulation corrupted by servos position resolution (B)



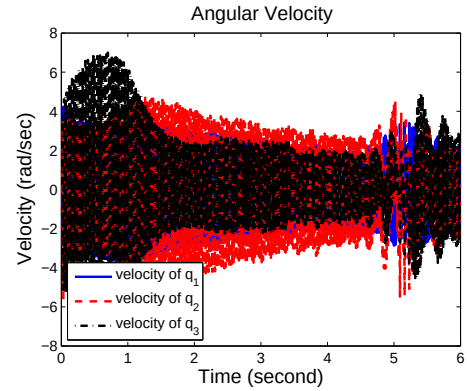
(a) (C) T_s=0.01 sec



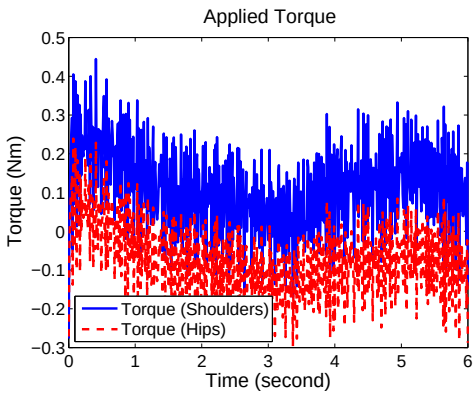
(b) (D) T_s=0.01 sec



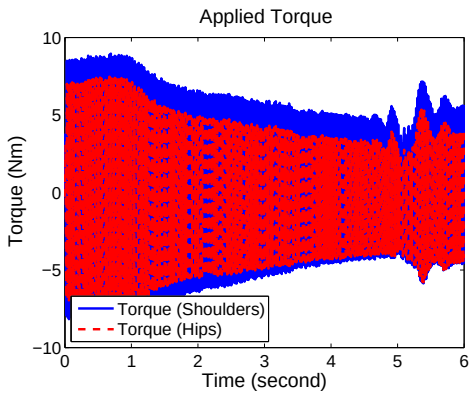
(c) (C) T_s=0.01 sec



(d) (D) T_s=0.01 sec

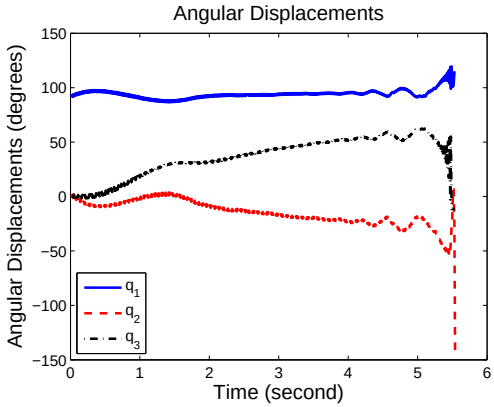


(e) (C) T_s=0.01 sec

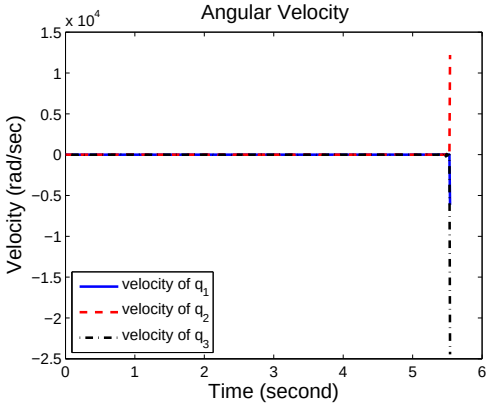


(f) (D) T_s=0.01 sec

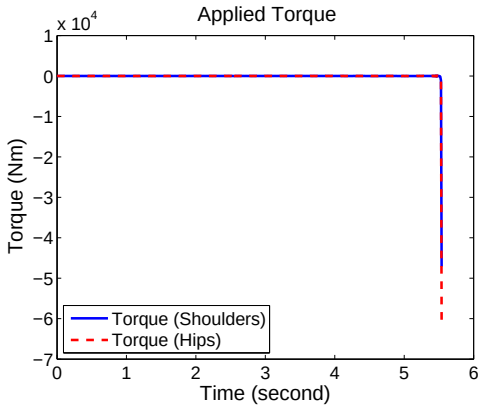
Figure 6.6: Simulation after adding gyroscope resolution (C) and simulation with dead-zone response of the servos (D)



(a) (E) T_s=0.01 sec



(b) (E) T_s=0.01 sec



(c) (E) T_s=0.01 sec

Figure 6.7: Simulation after adding friction coefficient between hands and high bar (E)

Chapter 7

Conclusions and future work

The work developed in this thesis in collaboration with Robosavvy Ltd was the basis for the creation of the Humanoid Robotics Laboratory of the IDMEC-Center of Intelligent Systems at Instituto Superior Técnico in (http://humanoids.dem.ist.utl.pt/). Consequently, this work aimed the creation of the foundations for future developments in humanoid robots. It is important to note that the process of modeling the multi-body structure of a humanoid robot, either for the purpose of doing a hand-stand on a high bar or for generating a stable walking motion, is a very complex one. One must deal with the formulation and solution of highly nonlinear dynamics equations of a very large size since a standard humanoid robot has typically 18 servos.

During this project, the following topics have been successfully achieved:

1. The construction of a serial protocol communication in real time between the PC and the humanoid robot, using Matlab/Simulink® Real-Time Workshop Toolbox®.
2. The external physics parameter identification of the humanoid structure, such as the mass, inertia tensor and centers of gravity of its main parts.
3. The dynamic analysis and identification of the internal behavior of the servo-actuators.
4. The development of a simulator for the humanoid doing a handstand on bar using virtual reality as animation.
5. The two sets of the humanoid 3D CAD drawing and its constituents. One set is detailed, resembling the reality pieces for mechanical analysis, while a less detailed one with precise real measurements is used in virtual reality animation.
6. The dynamics modeling and simulation using SimMechanics® and Virtual Reality Toolbox® for the equilibrium phase of the humanoid doing a hand-stand on a high-bar.

7. The deduction of the equations of motions for a n-link inverted pendulum and its linearization along the vertical unstable position, and ways to control a n-link inverted pendulum with eccentrics masses using a linear quadratic regulator.

8. The simulation of a linear quadratic regulator controller for the equilibrium phase of the humanoid doing a hand-stand on a high bar.

In terms of control and simulation, the humanoid was treated as a three body serial chain in an inverted pendulum configuration. The system is underactuated, being the motion of the legs and torso prescribed in order to stabilize the full body of the humanoid above the high bar. Optimal control methodologies were explored, being the Linear Quadratic Regulator (LQR) adopted in this thesis. This strategy was successfully applied in the stabilization of the humanoid on a high-bar although only in simulation. In fact, the real-time implementation of this controller proved to be unfeasible due to the high nonlinearities present on the servo dynamics, namely their dead-zone.

In this project a LQR controller was implemented to stabilize the humanoid robot on a high bar in simulation. This controller was shown not to be able to handle the nonlinearities present on the servos, and hence a new type of controllers must be developed and tested. The sliding mode control is seen at this moment a good alternative since it can handle uncertainty in a robust way (Utkin, 1992; Lee and Coverstone-Carroll, 1998; Qian et al., 2007). Nevertheless, intelligent control should be considered as well due to its new advances and results in controlling an underactuated inverted pendulum (Qian et al., 2006; Wu et al., 2007).

The dynamics of the humanoid robot used by the controller neglected the servo dynamics, despite their transfer functions have been accurately identified. Therefore these transfer functions should be considered in a future implementation of the controller.

The problem of stabilizing a walking motion is much more complex than that of stabilizing a hand-stand on a high-bar. At each step there are impact forces and transient mechanical constraints, and the high model size reduction of the former case may no longer be performed. Thus, nonlinear control approaches must be explored. A stable walking gait controller is the basis for more complex motions such as running and jumping, and this constitutes the main path for future work.

Bibliography

Asano, F. and Luo, Z.-W. (2007). "Dynamic analyses of underactuated virtual passive dynamic walking", *Robotics and Automation, 2007 IEEE International Conference on*, pp. 3210–3217.

Aurelie, D., Benjamin, M., Eric, B., Erik, H. and Jacob, R. (2006). Pendubot, *Technical report*, KTU.

Ayala Botto, M. (2003a). *Controlo de Sistemas - Acetatos das Aulas Teóricas*, Lisboa, Instituto Superior Técnico, AEIST.

Ayala Botto, M. (2003b). *Identificação de Sistemas - Acetatos das Aulas Teóricas*, Lisboa, Instituto Superior Técnico, AEIST.

Ayala Botto, M. (2004). *Controlo Não Linear - Acetatos das Aulas Teóricas*, Lisboa, Instituto Superior Técnico, AEIST.

Ayala Botto, M. (2006). *Controlo Óptimo - Acetatos das Aulas Teóricas*, Lisboa, Instituto Superior Técnico, AEIST.

Brown, S. C. and Passino, K. M. (1997). "Intelligent control for an acrobot", *Journal of Intelligent and Robotic Systems*, **18**(3), 209–248.

Bryant, R., Griffiths, P. and Grossman, D. (2003). *Exterior Differential Systems and Euler-Lagrange Partial Differential Equations*, University of Chicago Press.

Chestnutt, J., Lau, M., Cheung, G., Kuffner, J., Hodgins, J. and Kanade, T. (2005). "Footstep planning for the honda ASIMO humanoid", *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 629–634.

Collins, S., Ruina, A., Tedrake, R. and Wisse, M. (2005). "Efficient bipedal robots based on passive-dynamic walkers", *Science Magazine*, **307**, 1082–1085.

Hirukawa, H., Hattori, S., Kajita, S., Harada, K., Kaneko, K., Kanehiro, F., Morisawa, M. and Nakaoka, S. (2007). "A pattern generator of humanoid robots walking on a rough terrain", *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2181–2187.

Kim, J., Park, I. and Oh, J. (2007). "Walking control algorithm of biped humanoid robot on uneven and inclined floor", *Journal of Intelligent and Robotic Systems*, **48**(4), 457–484.

Lancaster, P. and Rodman, L. (1995). *Algebraic Riccati Equations*, Oxford University Press.

Lee, K., Coates, S. and Coverstone-Carroll, V. (1997). "Variable structure control applied to underactuated robots", *Robotica*, **15**(3), 313–318.

Lee, K. and Coverstone-Carroll, V. (1998). "Control algorithms for stabilizing underactuated robots", *Journal of Robotic Systems*, **15**(12), 681–697.

Ljung, L. (1987). *System Identification: theory for the user*, Prentice-Hall.

Maia, N. M. M. (2000). *Introdução á dinâmica analítica*, Lisboa, IST Press.

Miron, R., Hrimiuc, D., Shimada, H. and Sabau, S. V. (2002). *The Geometry of Hamilton and Lagrange Spaces*, Kluwer Academic Publishers.

Ogata, K. (2002). *Modern Control Engineering*, Vol. Fourth Edition, University of Minnesota, Prentice Hall.

Popovic, M.;and Hofmann, A. and Herr, H. (2004). "Angular momentum regulation during human walking: biomechanics and control", *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, **3**, 2405–2411.

Qian, D., Yi, J. and Zhao, D. (2006). "Direct adaptive control for underactuated mechatronic systems using fuzzy systems and neural networks: A pendubot case", *Electrical and Computer Engineering, 2006. CCECE '06. Canadian Conference on*, pp. 1490–1493.

Qian, D., Yi, J. and Zhao, D. (2007). "Hierarchical sliding mode control to swing up a pendubot", *American Control Conference, 2007. ACC '07*, pp. 5254–5259.

Ramamoorthy, S. (2007). *Task Encoding, Motion Planning and Intelligent Control using Qualitative Models*, PhD thesis, The University of Texas at Austin.

Ribeiro, M. I. (2002). *Análise de Sistemas Lineares*, Vol. 1 and 2, Lisboa, IST Press.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N. and Fujimura, K. (2002). "The intelligent ASIMO: System overview and integration", *Intelligent Robots and System, 2002. IEEE/RSJ International Conference on*, **3**, 2478– 2483.

Sockol, M., Raichlen, D. and Pontzer, H. (2007). "Chimpanzee locomotor energetics and the origin of human bipedalism", *Proceedings of the National Academy of Sciences*, **104**(30), 12265–12269.

Spong, M. W. (1994). "Partial feedback linearization of underactuated mechanical systems", *Intelligent Robots and Systems '94. 'Advanced Robotic Systems and the Real World', IROS '94. Proceedings of the IEEE/RSJ/GI International Conference on*, **1**, 314–321.

Spong, M. W. and Block, D. J. (1995). "The pendubot: a mechatronic system for control research and education", *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, **1**, 555–556.

Spong, M. W. and Vidyasagar, M. (1989). *Robots Dynamics and Control*, New York, John Wiley and Sons, Inc.

Takashima, S. (1989). *Robots Dynamics and Control*, Osaka, Japan, IEEE Workshop on Inteligent Robots and Systems, IROS'91.

Takenaka, T. (2006). "The control system for the honda humanoid robot", *Age and Ageing*, **35**(1), ii24–ii26.

Utkin, V. I. (1992). *Sliding Modes in Control and Optimization*, New York, Springer-Verlag.

White, W.N., J., Niemann, D. and Lynch, P. (1989). "The presentation of lagranges equations in introductory robotics courses", *Education, IEEE Transactions on*, **32**(1), 39–46.

Wolpert, D., Ghahramani, Z. and Flanagan, J. (2001). "Perspectives and problems in motor learning", *Trends in Cognitive Sciences*, **5**(11), 487–494.

Wu, C.-J., Lee, T.-L., Fu, Y.-Y. and Lai, L.-C. (2007). "Auto-tuning fuzzy pid control of a pendubot system", *Mechatronics, ICM2007 4th IEEE International Conference on*, pp. 1–6.

Appendix A

C-MEX S-Function code for real-time communication

The C-MEX S-Function code used for establishing the protocol communication between the PC using Matlab/Simulink® and the humanoid robot is described in this appendix. The code is splited into 5 simple parts for full understanding.

The first part of the code is for the declarations, just as setting the COM port (Figure A.1).

The second part of the program initializes the Input and Output signal of the S-function and RS232 protocol communication at the baudrate (Figure A.2).

In the third part (Figure A.3), the read function from CM5 is presented. Two bytes, corresponding to the actual position of the servo, are read. A system of flags were used in order to prevent possible receiving errors.

The fourth part is the principal block (Figure A.4). It receives the desired final position for the servo from Simulink and sends it as two bytes for the AX-12 throughout CM5. After sending the bytes, it receives the actual position of the servos, sending to the CM5 first a confirmation that it is ready to start reading the values.

The last part stands for the conclusion of the S-Function (Figure A.5).

```
/*PC_code
 *
 *This function writes a value of two bytes to the CM5 and reads
 *and reads from the CM5 also a value of two bytes
 *
 */

#define S_FUNCTION_NAME  PC_code
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "stdlib.h"
#include "stdio.h"
#include "math.h"
#include "conio.h" /* inp(), outp(), inpw(), outpw() */
#define inportb(num) ((unsigned char) inp( num ))
#define outportb(num, value) ((void) outp( num, value ))

//----- Declarations for use -----

#define WRITEERROR_BYTE -1  //error when writing 2 bytes
#define WRITEERROR_CHAR -2  //error when writing 1 byte
#define READERROR_BYTE -11  //error when reading 1 byte

#define CM5_SERVOS 1        //number of servos that sfunction receives
#define PORT1 0x3F8         //define COM1 as the port for communication
                            /* COM1 0x3F8   */
                            /* COM2 0x2F8   */
                            /* COM3 0x3E8   */
                            /* COM4 0x2E8   */

//----- global variables -----

short counter;              //counts nb of ticks between request and write or read a byte
short f;                    //flag turns if a byte was read or write


```

Figure A.1: Part 1 of the C-MEX S-function

```
37  static void mdlInitializeSizes(SimStruct *S)
38  {
39
40      if (!ssSetNumInputPorts(S, 1)) return;     //check if there is only 1 input signal
41      ssSetInputPortWidth(S, 0, 1);              //the input signal is a scalar
42      ssSetInputPortDirectFeedThrough(S, 0, 1);  //signal to be used in mdlOutputs
43      //ssSetInputPortRequiredContiguous(S, 0, 1);
44
45      if (!ssSetNumOutputPorts(S, 1)) return;    //check if there is only 1 output signal
46      ssSetOutputPortWidth(S,  0, 1);            //the output signal is a scalar
47
48      ssSetNumSampleTimes(S, 1);                 //number of sample times
49
50  #ifndef MATLAB_MEX_FILE
51
52      //----- set usart registers -----
53
54      outportb(PORT1 + 1 , 0);         // Turn off interrupts - Port1
55      outportb(PORT1 + 3 , 0x80);      // SET DLAB ON                              */
56      outportb(PORT1 + 0 , 0x02);      /* Set Baud rate - Divisor Latch Low Byte   */
57                                       /*         0x03 =  38,400 BPS               */
58                                       /*         0x01 = 115,200 BPS               */
59                                       /*         0x02 =  57,600 BPS               */
60                                       /*         0x06 =  19,200 BPS               */
61                                       /*         0x0C =   9,600 BPS               */
62                                       /*         0x18 =   4,800 BPS               */
63                                       /*         0x30 =   2,400 BPS               */
64      outportb(PORT1 + 1 , 0x00);      // Set Baud rate - Divisor Latch High Byte
65      outportb(PORT1 + 3 , 0x03);      // 8 Bits, No Parity, 1 Stop Bit
66      // FIFO Control Register used to be 0xCF ie: enabled with 14 bytes of fifo*/
67      outportb(PORT1 + 2 , 0x00);
68
69  #endif
70  }
71
72  static void mdlInitializeSampleTimes(SimStruct *S)
73  {
74      ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);
75      ssSetOffsetTime(S, 0, 0.0);
76  }
77
```

Figure A.2: Part 2 of the C-MEX S-function

```
78  /* Function: serial_read_ushort
79   * Abstract:
80   *    In this function, PC reads from CM5 2 bytes
81   */
82
83  unsigned short serial_read_ushort () {
84      unsigned char lo;
85      unsigned char hi;
86
87  #ifndef MATLAB_MEX_FILE
88
89      //----- read 1st byte from CM5:
90      counter=0;
91      while(counter<10000){
92          if(inportb(PORT1 + 5)&1) {  //if byte ready to read
93              lo = inportb(PORT1);    //read the byte
94              f=1;                    //flag=1
95              break;                  //exit
96          }t
97          counter++;
98      }
99
100     //----- read 2nd byte from CM5 if 1st byte was read:
101     if(f==1){
102         counter=0;
103         while(counter<10000){
104             if(inportb(PORT1 + 5)&1) {  //if byte ready to read
105                 hi = inportb(PORT1);    //read the byte
106                 f=2;                    //flag=2
107                 break;                  //exit
108             }
109             counter++;
110         }
111     }
112
113     //----- return the number if the 2 bytes were read or error otherwise
114     if(f==2)
115     {
116         return ( ((unsigned short)lo) +
117             (((unsigned short)hi)<<8)   );
118     }
119     else
120     {
121            return READERROR_BYTE;
122     }
123
124     return ( ((unsigned short)lo) + (((unsigned short)hi)<<8)   );
125 #else
126     static unsigned short simulation=0;
127     return (simulation++);
128 #endif
129 }
130
```

Figure A.3: Part 3 of the C-MEX S-function

```
132 /* Function: mdlOutputs
133  * Abstract:
134  *    In this function, you compute the outputs of your S-function
135  *    block.
136  */
137
138 static void mdlOutputs(SimStruct *S, int_T tid)
139 {
140     unsigned short v;                                                    // input value
141     real_T *simulink_input = (real_T*) ssGetInputPortSignal(S,0);        // values -1..1
142     real_T *simulink_output = (real_T *) ssGetOutputPortSignal(S,0);     // scalar
143     unsigned char out_buf[2];   //vector that contains the lo and hi byte of the input value
144
145     v=(unsigned short)(simulink_input[0]);
146
147     out_buf[0] = v&0xff;
148     out_buf[1] = v>>8;
149
150
151 #ifndef MATLAB_MEX_FILE
152
153 //[][][][][][] write 2 bytes to CM5: [][][][][][]
154
155     //----- write 1st byte to CM5:
156     counter=0;
157     while(counter<10000){
158         if(inportb(PORT1 + 5)&(1<<5)){          //if byte ready to write
159             outportb(PORT1, out_buf[0]);        //write byte
160             f=1;                                //flag=1
161             break;                              //exit
162         }
163         counter++;
164     }
165
166     //----- write 2nd byte to CM5 if 1st byte was write:
167     if(f==1){
168         counter=0;
169         while(counter<10000){
170             if(inportb(PORT1 + 5)&(1<<5)){      //if byte ready to write
171                 outportb(PORT1, out_buf[1]);    //write byte
172                 f=2;                            //flag=2
173                 break;                          //exit
174             }
175             counter++;
176         }
177     }
178
179
180 //[][][][][][] read 2 bytes from CM5: [][][][][][]
181
182     //----- if the 2 bytes ere sent nicely then
183     //----- write a character to CM5, meaning that the PC wants to read 2 bytes
184     if(f==2) {
185         counter=0;
186         f=0;
187         while(counter<10000){
188             if(inportb(PORT1 + 5)&(1<<5)) {     //if byte ready to write
189                 outportb(PORT1, 'c');           //write just a char
190                 f=1;                            //flag=1
191                 break;                          //exit
192             }
193         }
194
195         //----- call function to read 2 bytes from CM5 if char was write
196         if(f==1){
197             v = serial_read_ushort();
198         }
199         else{
200             v=WRITEERROR_CHAR;
201         }
202         simulink_output[0] = v;
203     }
204     else{
205         simulink_output[0] = WRITEERROR_BYTE;
206     }
207 #endif
208
209 }
210
```

Figure A.4: Part 4 of the C-MEX S-function

```
212  /* Function: mdlTerminate
213   * Abstract:
214   *    In this function, you should perform any actions that are necessary
215   *    at the termination of a simulation.  For example, if memory was
216   *    allocated in mdlStart, this is the place to free it.
217   */
218  static void mdlTerminate(SimStruct *S)
219  {
220
221  }
222
223  /*=============================*
224   * Required S-function trailer *
225   *=============================*/
226
227  #ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
228  #include "simulink.c"      /* MEX-file interface mechanism */
229  #else
230  #include "cg_sfun.h"       /* Code generation registration function */
231  #endif
232
```

Figure A.5: Part 5 of the C-MEX S-function

Appendix B

Protocol communication C code for PC and humanoid robot

In this appendix, the required C code for establishing a serial protocol communication between PC and a servo of the humanoid robot is presented. The objective of this protocol is to send a reference angular velocity to servo ever 0.01s and to receive from it a roll of different data, such as current angular position, angular velocity, DC current, voltage and temperature. Servo angular velocity is also estimated from current position using an online filter. The required steps for this protocol implementation are:

1. C code for the microprocessor Atmega128. Figure B.1 shows the main function to be used inside the "example.c" code of Robotis.
2. C-MEX S-Function code to use in the Matlab/Simulink®. In Appendix A, it is shown the full C-MEX S-Function developed for sending and receiving position from a servo. The same C-MEX S-Function is used in this case with minor changes shown in the Figures B.2 and B.3.
3. Simulink block diagram shown in Figure B.4. This diagram also compares the response of the estimated current speed of a servo (online derivating of the current position (Figure B.6)) with the output speed given by the transfer function. It also calculate the reference position by integrating the reference speed (Figure B.5).

```
(. . .)
int main(void) {
  byte bCount,bID, bTxPacketLength,bRxPacketLength;

  PortInitialize(); //Port In/Out Direction Definition
  RS485_RXD; //Set RS485 Direction to Input State.

  //RS485 Initializing(RxInterrupt)
  SerialInitialize(SERIAL_PORT0,1,RX_INTERRUPT);

  //RS232 Initializing(None Interrupt)
  SerialInitialize(SERIAL_PORT1,DEFAULT_BAUD_RATE,0);

  gbRxBufferReadPointer = gbRxBufferWritePointer = 0;  //RS485 RxBuffer Clearing.

  sei();  //Enable Interrupt -- Compiler Function

/***************************************
 *         code to be inserted         *
 ***************************************/

  //servo to be used
  bID = 1;

  //goal position splited in lo and hi byte
  byte lo, hi;

  //array of servo data to be sent to PC
  byte send[10];

  while(1)
  {

    //read from MATLAB the reference speed
    lo=RxD8();
    hi=RxD8();

    //send the goal position to the servo
    gbpParameter[0] = P_GOAL_SPEED_L;     //Address of Firmware Version
    gbpParameter[1] = lo;                  //Writing Data P_GOAL_SPEED_L
    gbpParameter[2] = hi;                  //Writing Data P_GOAL_SPEED_H
    bTxPacketLength = TxPacket(bID,INST_WRITE,3);
    bRxPacketLength = RxPacket(DEFAULT_RETURN_PACKET_SIZE);


    //get present position
    gbpParameter[0] = P_PRESENT_POSITION_L; //Address 36 of present position LSB
    gbpParameter[1] = 2;                    // Read 2 bytes starting at address 36
    TxPacket(bID,INST_READ,2);              // 2 means 2 bytes in parameter array
    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
    send[0]=gbpRxBuffer[5];
    send[1]=gbpRxBuffer[6];

    //get present speed
    gbpParameter[0] = P_PRESENT_SPEED_L; //Address 38 of present speed LSB
    gbpParameter[1] = 2;                    // Read 2 bytes starting at address 38
    TxPacket(bID,INST_READ,2);              // 2 means 2 bytes in parameter array
    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
    send[2]=gbpRxBuffer[5];
    send[3]=gbpRxBuffer[6];

    //get present dc current
    gbpParameter[0] = P_PRESENT_LOAD_L; //Address 40 of present load LSB
    gbpParameter[1] = 2;                    // Read 2 bytes starting at address 40
    TxPacket(bID,INST_READ,2);              // 2 means 2 bytes in parameter array
    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
    send[4]=gbpRxBuffer[5];
    send[5]=gbpRxBuffer[6];

    //get present voltage
    gbpParameter[0] = P_PRESENT_VOLTAGE; //Address 42 of present volatge
    gbpParameter[1] = 1;                    // Read 1 byte starting at address 42
    TxPacket(bID,INST_READ,1);              // 1 means 1 byte in parameter array
    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
    send[6]=gbpRxBuffer[5];
    send[7]=0x00;

    //get present temperature
    gbpParameter[0] = P_PRESENT_TEMPERATURE; //Address 43 of present temperature
    gbpParameter[1] = 1;                    // Read 1 bytes starting at address 43
    TxPacket(bID,INST_READ,1);              // 1 means 1 byte in parameter array
    RxPacket(DEFAULT_RETURN_PACKET_SIZE+gbpParameter[1]);
    send[8]=gbpRxBuffer[5];
    send[9]=0x00;


    //for each instruction to be sent read first 1 byte from pc.
    RxD8();
    TxD81(send[0]);
    TxD81(send[1]);
    RxD8();
    TxD81(send[2]);
    TxD81(send[3]);
    RxD8();
    TxD81(send[4]);
    TxD81(send[5]);
    RxD8();
    TxD81(send[6]);
    TxD81(send[7]);
    RxD8();
    TxD81(send[8]);
    TxD81(send[9]);
  }

/****************************
 *         end code         *
 ****************************/

}
(. . .)
```

Figure B.1: Microprocessor C code for PC-Servo protocol

```
(...)
static void mdlInitializeSizes(SimStruct *S) {

   //check if there is only 1 input signal
   if (!ssSetNumInputPorts(S, 1)) return;
   //the input signal is a scalar
   ssSetInputPortWidth(S, 0, 1); //Reference Speed
   //signal to be used in mdlOutputs
   ssSetInputPortDirectFeedThrough(S, 0, 1);
   //ssSetInputPortRequiredContiguous(S, 0, 1);

   //check if there is five 5 output signal
   if (!ssSetNumOutputPorts(S, 5)) return;
   //the output signals are scalar
   ssSetOutputPortWidth(S,  0, 1); //Servo Position
   ssSetOutputPortWidth(S,  1, 1); //Servo Speed
   ssSetOutputPortWidth(S,  2, 1); //Servo DC current
   ssSetOutputPortWidth(S,  3, 1); //Servo Voltage
   ssSetOutputPortWidth(S,  4, 1); //Servo Temperature
   //number of sample times
   ssSetNumSampleTimes(S, 1);

(...)
```

Figure B.2: Part 1 of the C-MEX S-Function for PC-Servo protocol

```
(...)
//[][][][][][] read data from CM5: [][][][][][]

   //----- if the 2 bytes ere sent nicely then
   //----- during 5 times, write a character to CM5, and read 2 bytes; fo
   //---1- Servo present Position
   //---2- Servo present Speed
   //---3- Servo present DC current
   //---4- Servo Present Voltage
   //---5- Servo Present Temperature

   for (int i=0, i<5, i++){
    if(f==2) {
       counter=0;
       f=0;
       while(counter<10000){
           if(inportb(PORT1 + 5)&(1<<5)) {      //if byte ready to write
               outportb(PORT1, 'c');            //write just a char
               f=1;                             //flag=1
               break;                           //exit
           }
       }

       //----- call function to read 2 bytes from CM5 if char was write
       if(f==1){
           v = serial_read_ushort();
           f=2;
       }
       else{
           v=WRITEERROR_CHAR;
       }
       simulink_output[i] = v;
    }
    else{
       simulink_output[i] = WRITEERROR_BYTE;
    }
   }
#endif
}
(...)
```

Figure B.3: Part 2 of the C-MEX S-Function for PC-Servo protocol

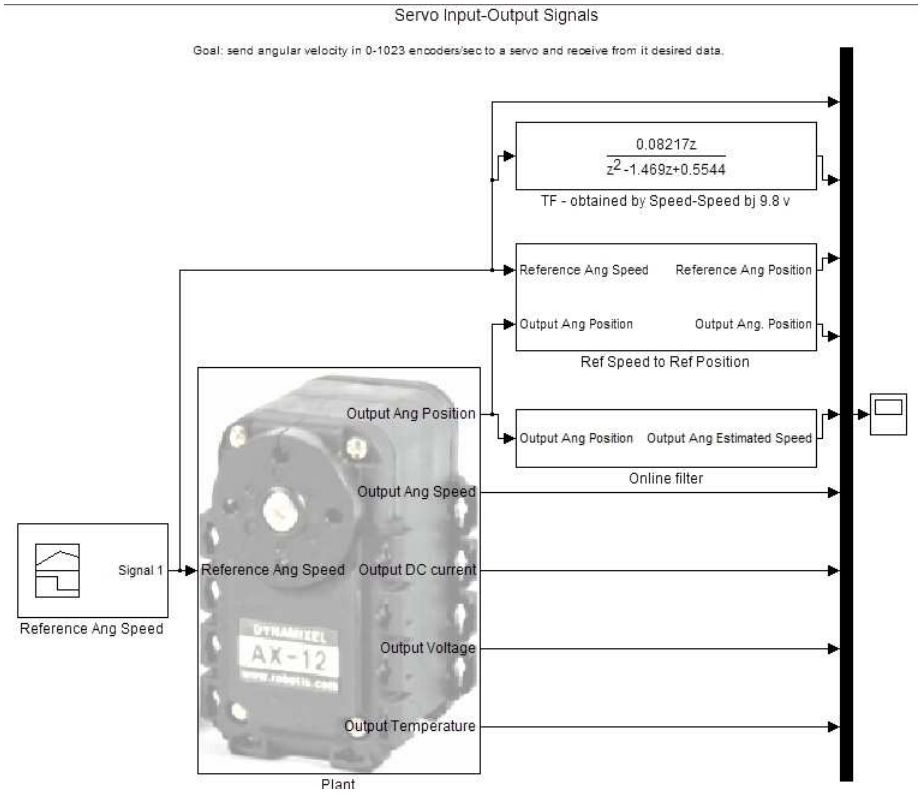


Figure B.4: Block Diagram for PC-Servo protocol

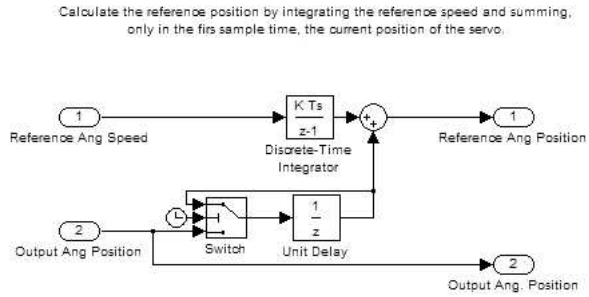


Figure B.5: Reference angular position block

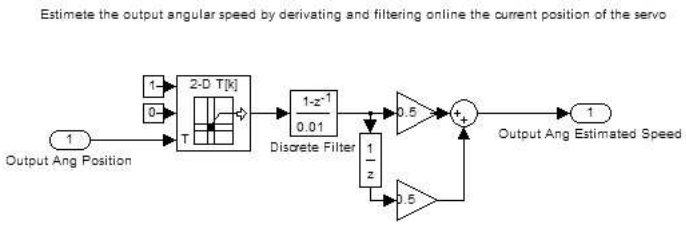


Figure B.6: Output estimated angular velocity using an online filter

Appendix C

Drawings and mechanical properties of the Walking humanoid

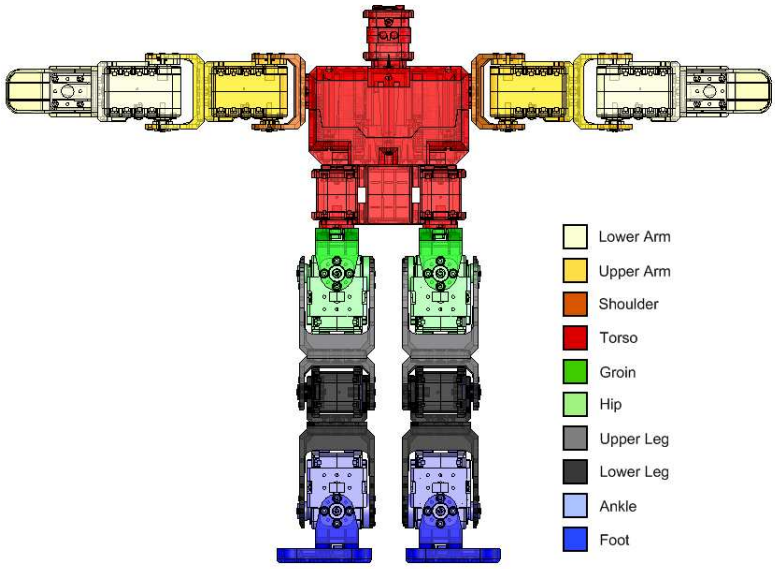
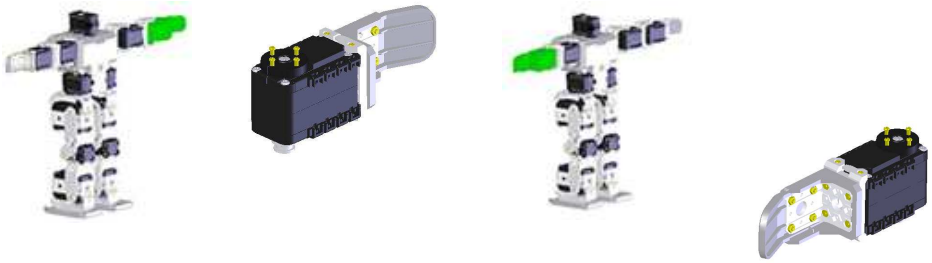


Figure C.1: Main Blocks of the Walking humanoid

Lower Arm



(a) Left Lower Arm (b) 3D CAD (c) Right Lower Arm (d) 3D CAD

Figure C.2: Lower Arm

	Left Lower Arm			Right Lower Arm		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
76.7	28460.9	34.2	3214.1	28460.9	34.2	−3214.1
	34.2	26727.7	508.9	34.21	26727.7	−508.9
	3214.1	508.9	5417.1	−3214.1	−508.9	5417.1

Table C.1: Lower Arm Mechanical Properties

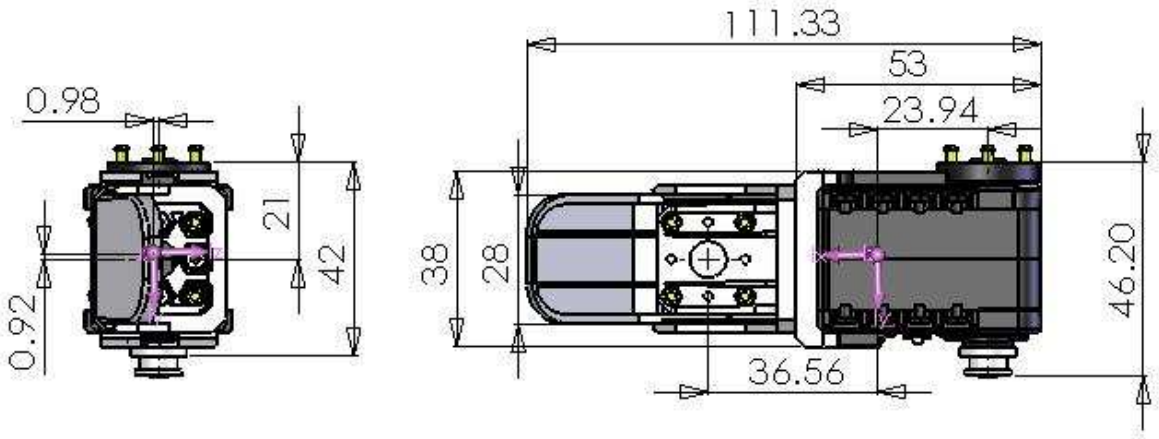
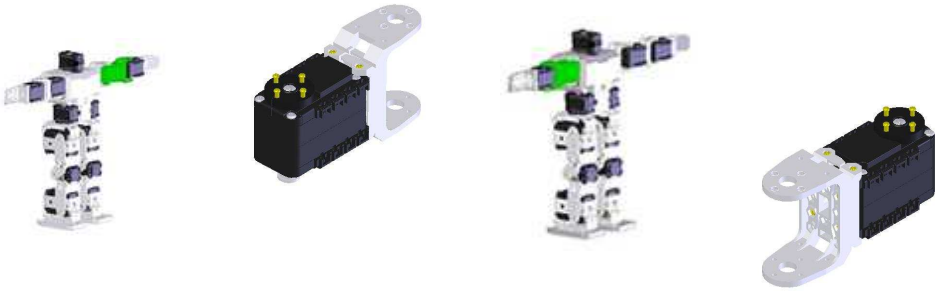


Figure C.3: Lower Arm 2D CAD

Upper Arm



(a) Left Upper Arm (b) 3D CAD (c) Right Upper Arm (d) 3D CAD

Figure C.4: Upper Arm

	Left Upper Arm			Right Upper Arm		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
76.5	23871.8	0.1	−0.3	23871.8	−0.1	−0.3
	0.1	19040.0	363.6	−0.1	19040.0	−363.6
	−0.3	363.6	7748.7	−0.3	−363.6	7748.7

Table C.2: Upper Arm Mechanical Properties

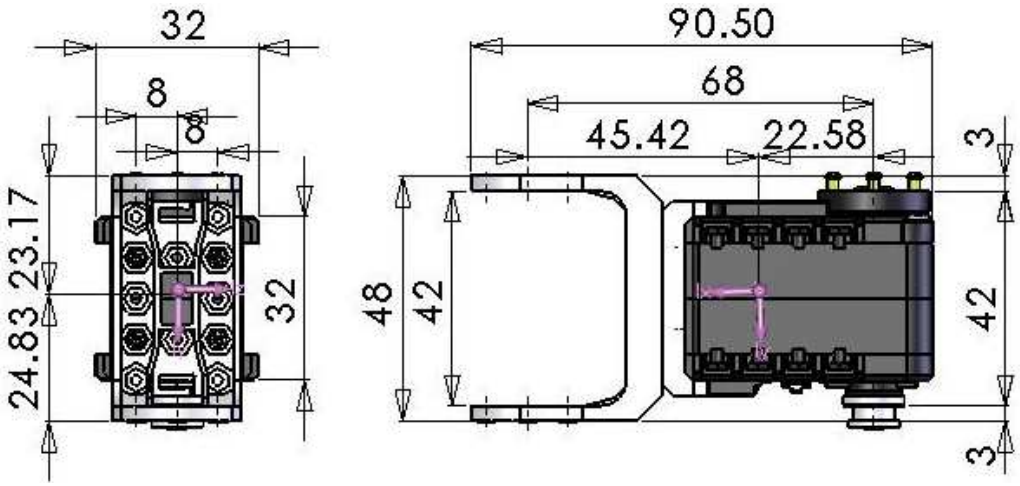


Figure C.5: Upper Arm 2D CAD

Shoulder

(a) Left shoulder (b) Right shoulder (c) 3D CAD

Figure C.6: Shoulder

	Left Shoulder			Right Shoulder		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
11.9	5168.6	0.0	−469.78	5168.6	0.0	469.78
	0.0	2205.0	0.0	0.0	2205.0	0.0
	−469.78	0.0	5295.7	469.78	0.0	5295.7

Table C.3: Shoulder Mechanical Properties

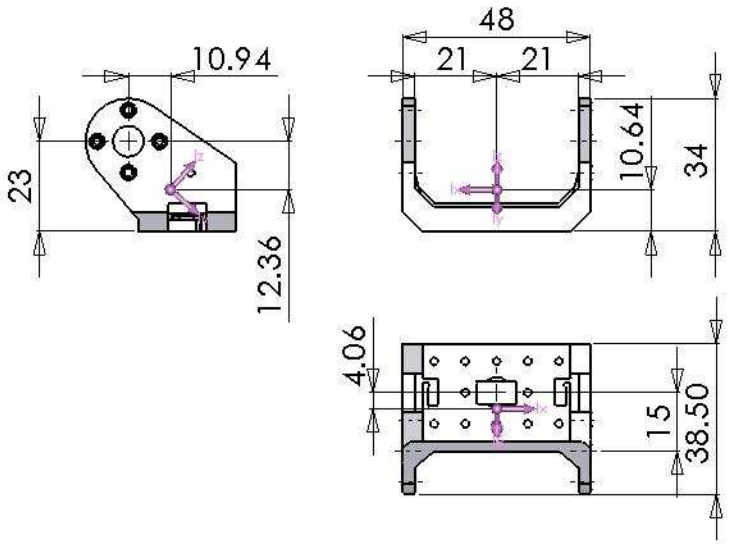
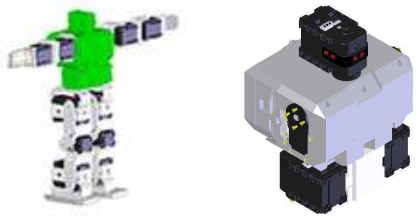


Figure C.7: Shoulder 2D CAD

Torso



(a) Left shoulder (b) Right CAD

Figure C.8: Torso

Torso	
Mass (g)	Inertia Tensor (gcm^2)
648.5	$\begin{bmatrix} 1043443.0 & -118883.3 & -5181.6 \\ -118883.3 & 775943.7 & 2242.5 \\ -5181.6 & 2242.5 & 844180.1 \end{bmatrix}$

Table C.4: Torso Mechanical Properties

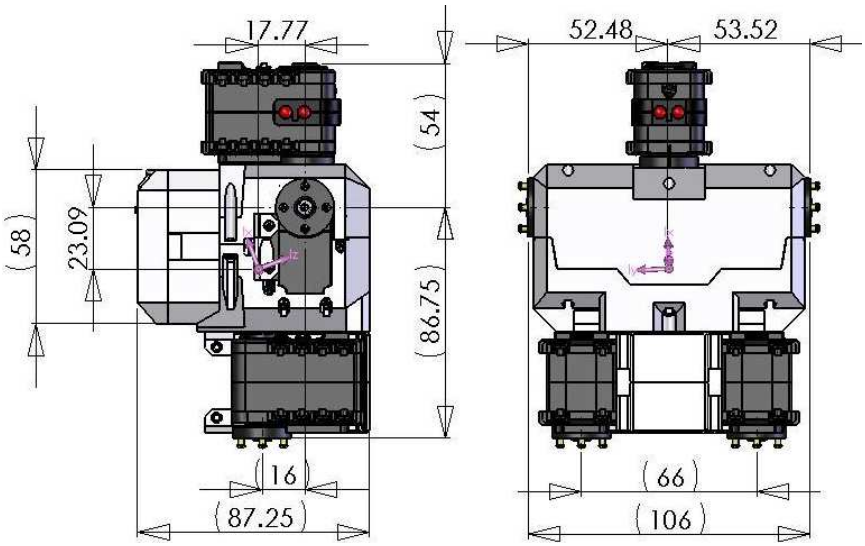


Figure C.9: Torso 2D CAD

Groin

(a) Left Groin (b) Right Groin (c) 3D CAD

Figure C.10: Groin

	Left Groin	Right Groin
Mass (g)	Inertia Tensor (gcm^2)	Inertia Tensor (gcm^2)
15.1	$\begin{bmatrix} 1957.9 & 78.2 & -0.5 \\ 78.2 & 9388.0 & 0.0 \\ -0.5 & 0.0 & 9832.1 \end{bmatrix}$	$\begin{bmatrix} 1957.9 & 78.2 & -0.5 \\ 78.2 & 9388.0 & 0.0 \\ -0.5 & 0.0 & 9832.1 \end{bmatrix}$

Table C.5: Groin Mechanical Properties

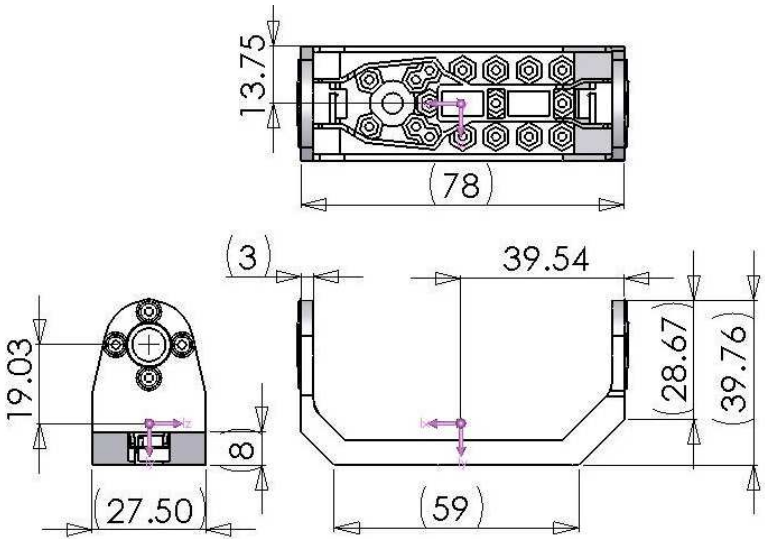
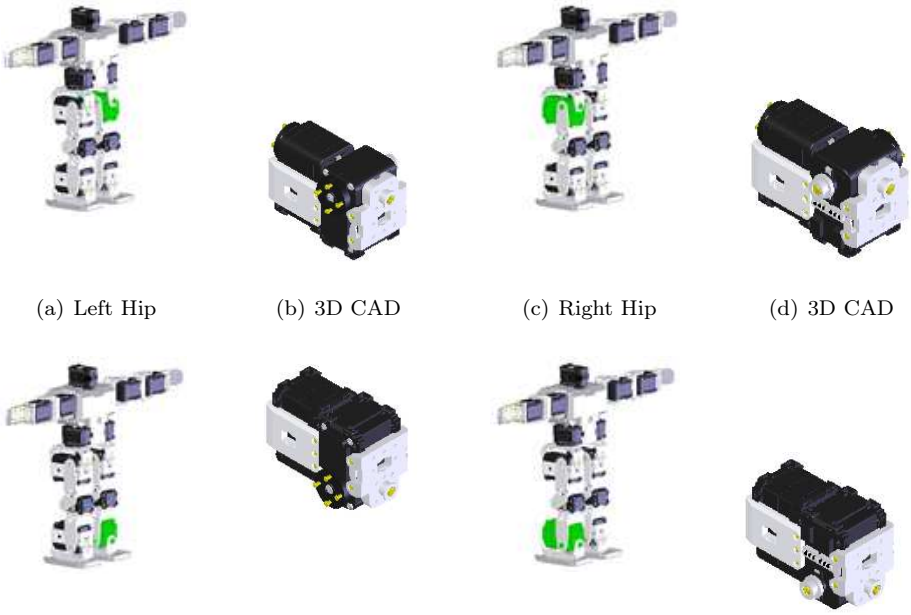


Figure C.11: Groin 2D CAD

Hip/Ankle



(a) Left Hip (b) 3D CAD (c) Right Hip (d) 3D CAD

(e) Left Ankle (f) 3D CAD (g) Right Ankle (h) 3D CAD

Figure C.12: Hip/Ankle

	Left Ankle			Right Ankle		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
137.9	11354.7	−1443.2	−1016.7	11354.7	−1443.2	−1016.7
	−1443.12	54852.3	−114.8	−1443.12	54852.3	−114.8
	−1016.7	−114.8	53390.0	−1016.7	−114.8	53390.0

Table C.6: Main Blocks Mechanical Properties

	Left Ankle			Right Ankle		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
137.9	11354.7	−1443.2	−1016.7	17688.8	−974.1	774.3
	−1443.12	54852.3	−114.8	−974.1	59738.1	−979.4
	−1016.7	−114.8	53390.0	774.3	−979.4	48519.1

Table C.7: Main Blocks Mechanical Properties

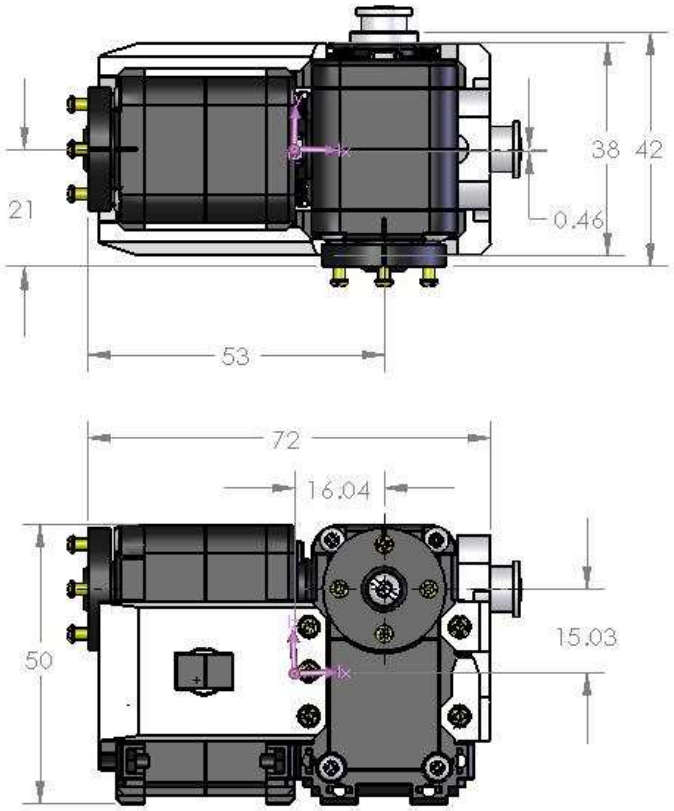


Figure C.13: Hip/Ankle 2D CAD

Upper Leg

(a) Left Upper Leg (b) Right Upper Leg (c) 3D CAD

Figure C.14: Upper Leg

	Left Upper Leg			Right Upper Leg		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
31.2	25374.1	1393.5	0.0	25374.1	1393.5	0.0
	1393.5	13828.1	0.0	1393.5	13828.1	0.0
	0.0	0.0	16927.5	0.0	0.0	16927.5

Table C.8: Upper Leg Mechanical Properties

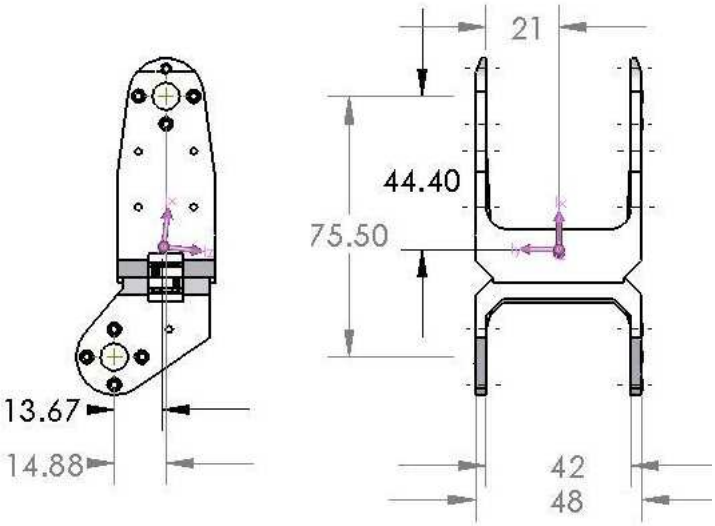
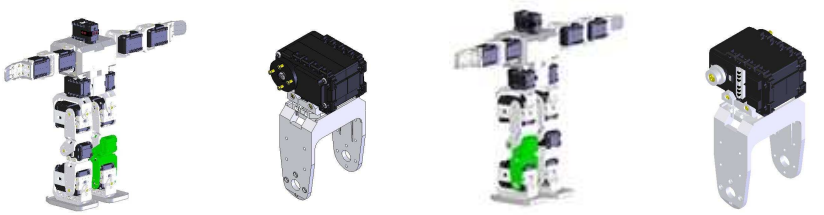


Figure C.15: Upper Leg 2D CAD

Lower Leg



(a) Left Lower Leg (b) 3D CAD (c) Right Lower Leg (d) 3D CAD

Figure C.16: Lower Leg

	Left Lower Leg			Right Lower Leg		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
86.6	43545.7	124.6	101.4	43545.7	124.6	-101.4
	124.6	13525.4	703.2	124.6	13525.4	-703.2
	101.4	703.2	37232.4	-101.4	-703.2	37232.4

Table C.9: Lower Leg Mechanical Properties

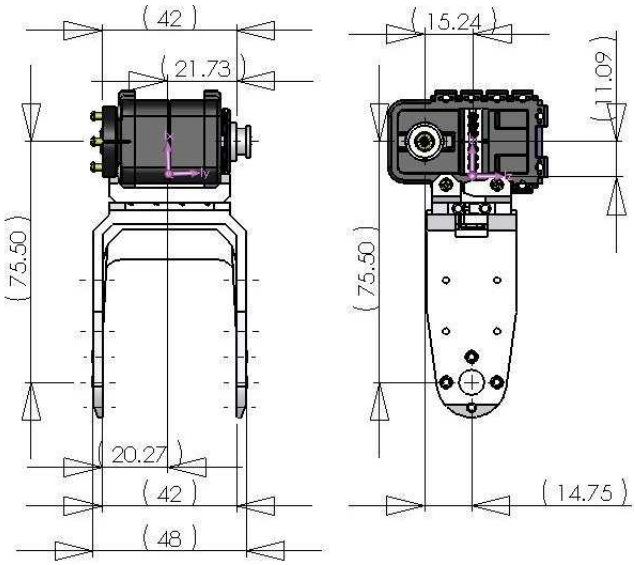
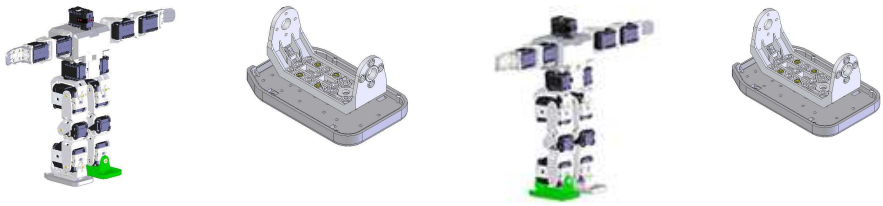


Figure C.17: Lower Leg 2D CAD

Foot



(a) Left Foot (b) 3D CAD (c) Right Foot (d) 3D CAD

Figure C.18: Foot

	Left Foot			Right Foot		
Mass (g)	Inertia Tensor (gcm^2)			Inertia Tensor (gcm^2)		
54.4	43545.7	124.6	101.4	43545.7	124.6	-101.4
	124.6	13525.4	703.2	124.6	13525.4	-703.2
	101.4	703.2	37232.4	-101.4	-703.2	37232.4

Table C.10: Foot Mechanical Properties

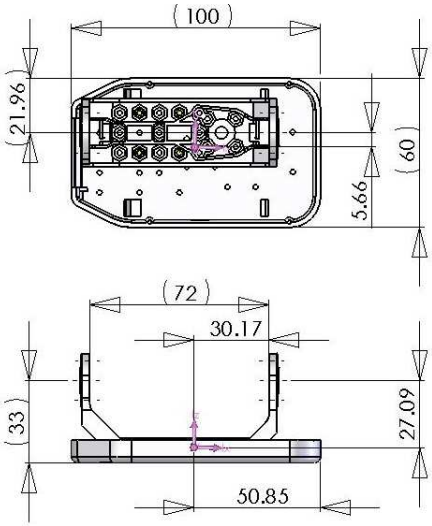


Figure C.19: Foot 2D CAD

Appendix D

Drawings and mechanical properties of the Gymnast Humanoid

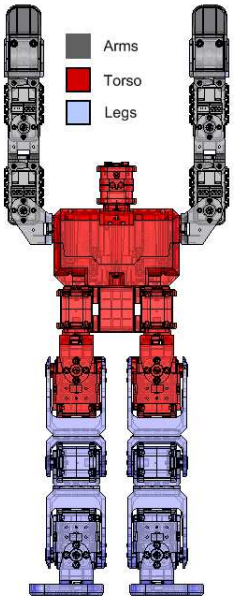
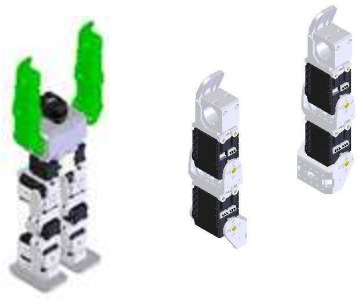


Figure D.1: Main Blocks of the Gymnast Humanoid

Arms



(a) Arms (b) 3D CAD

Figure D.2: Arms

Arms			
Mass (g)	Inertia Tensor (gcm^2)		
367.6	28479.93	59.0	-0.7
	59.0	21258.1	-1.3
	-0.7	-1.3	7890.7

Table D.1: Arms Mechanical Properties

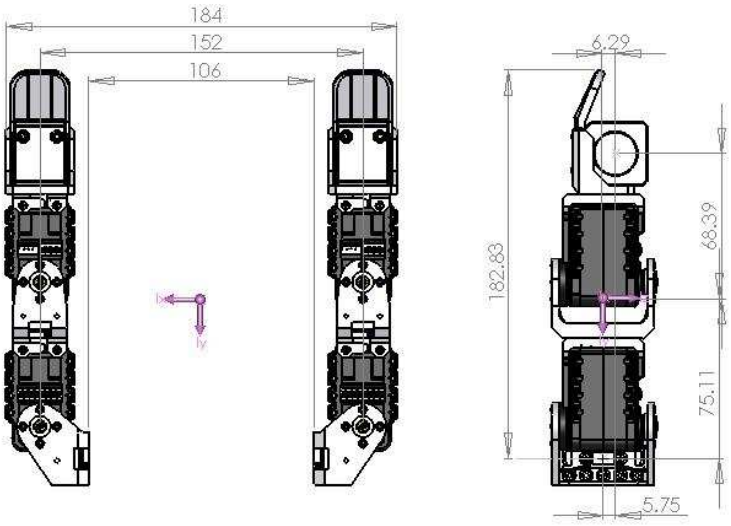
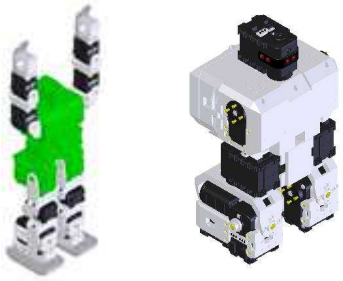


Figure D.3: Arms 2D CAD

Torso



(a) Arms and Torso (b) 3D CAD

Figure D.4: Torso

Arms	
Mass (g)	Inertia Tensor (gcm^2)
981.5	$\begin{bmatrix} 37029.8 & -1750.1 & -54.7 \\ -1750.1 & 12211.3 & 134.6 \\ -54.7 & 134.6 & 32898.6 \end{bmatrix}$

Table D.2: Torso Mechanical Properties

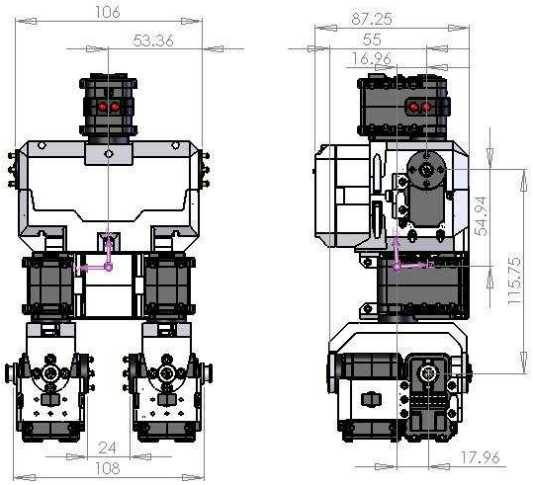
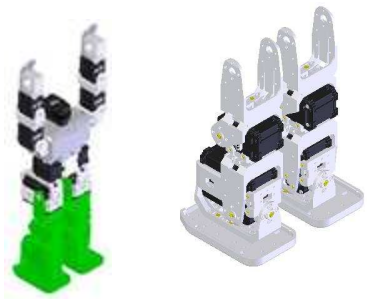


Figure D.5: Torso 2D CAD

Legs



(a) Arms and Torso (b) 3D CAD

Figure D.6: Legs

Legs			
Mass (g)	Inertia Tensor (gcm^2)		
576.4	16420.4	1329.6	0.1
	1329.6	9032.2	1.4
	0.1	1.4	11328.0

Table D.3: Legs Mechanical Properties

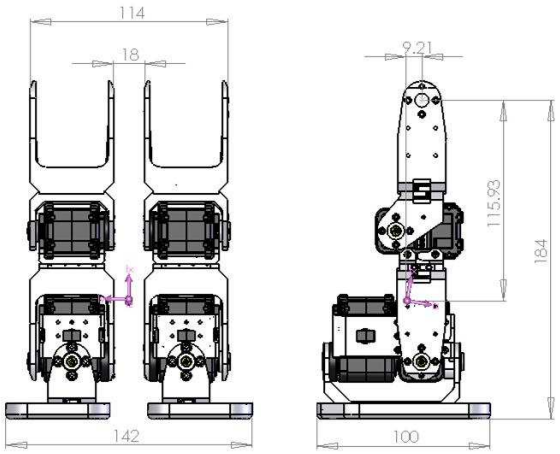


Figure D.7: Legs 2D CAD

Appendix E

Poles and zeros of the Gymnast Humanoid model

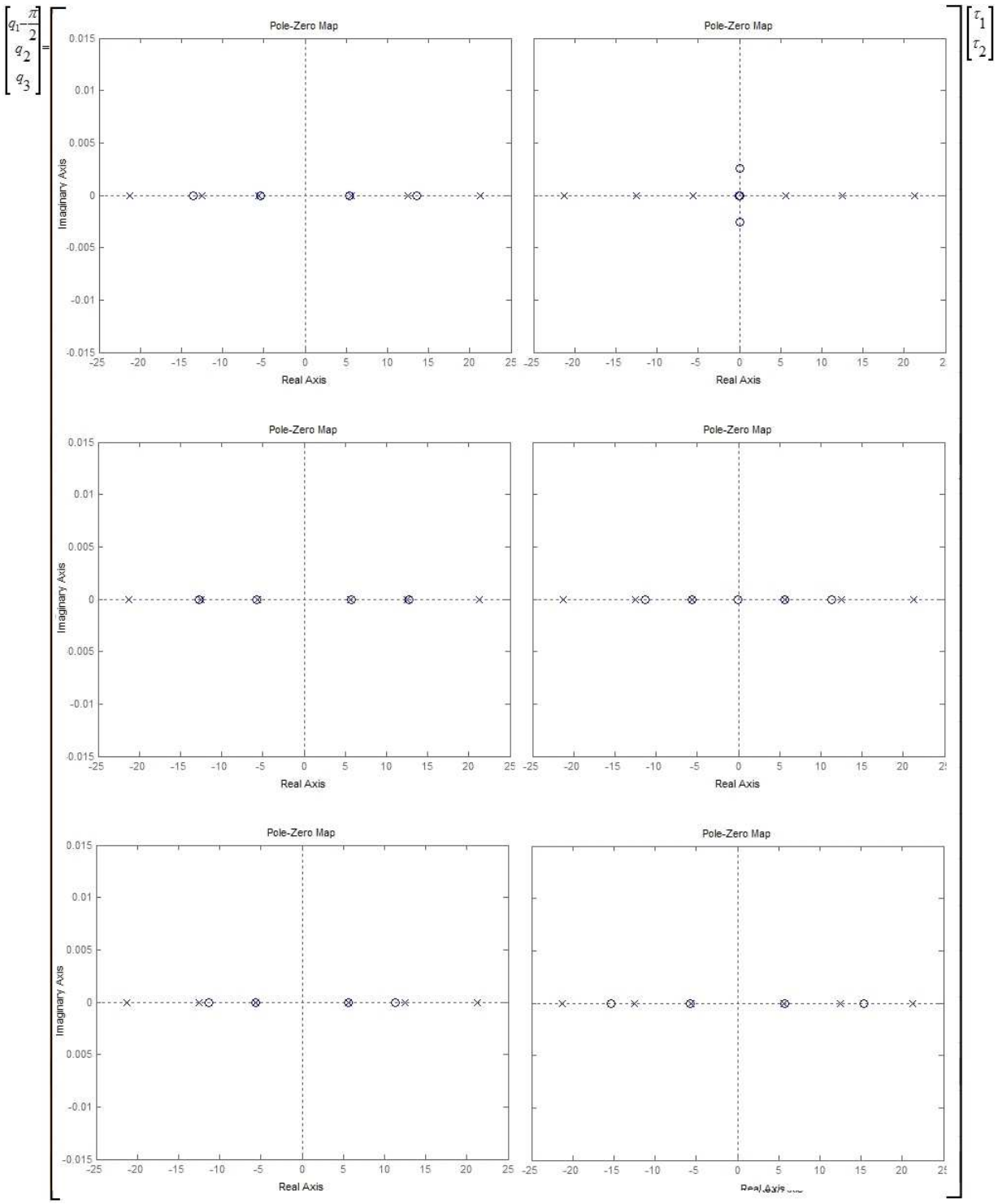


Figure E.1: Poles and zeros for the first 3 states $(q_1 - \frac{\pi}{2}, q_2, q_3)$

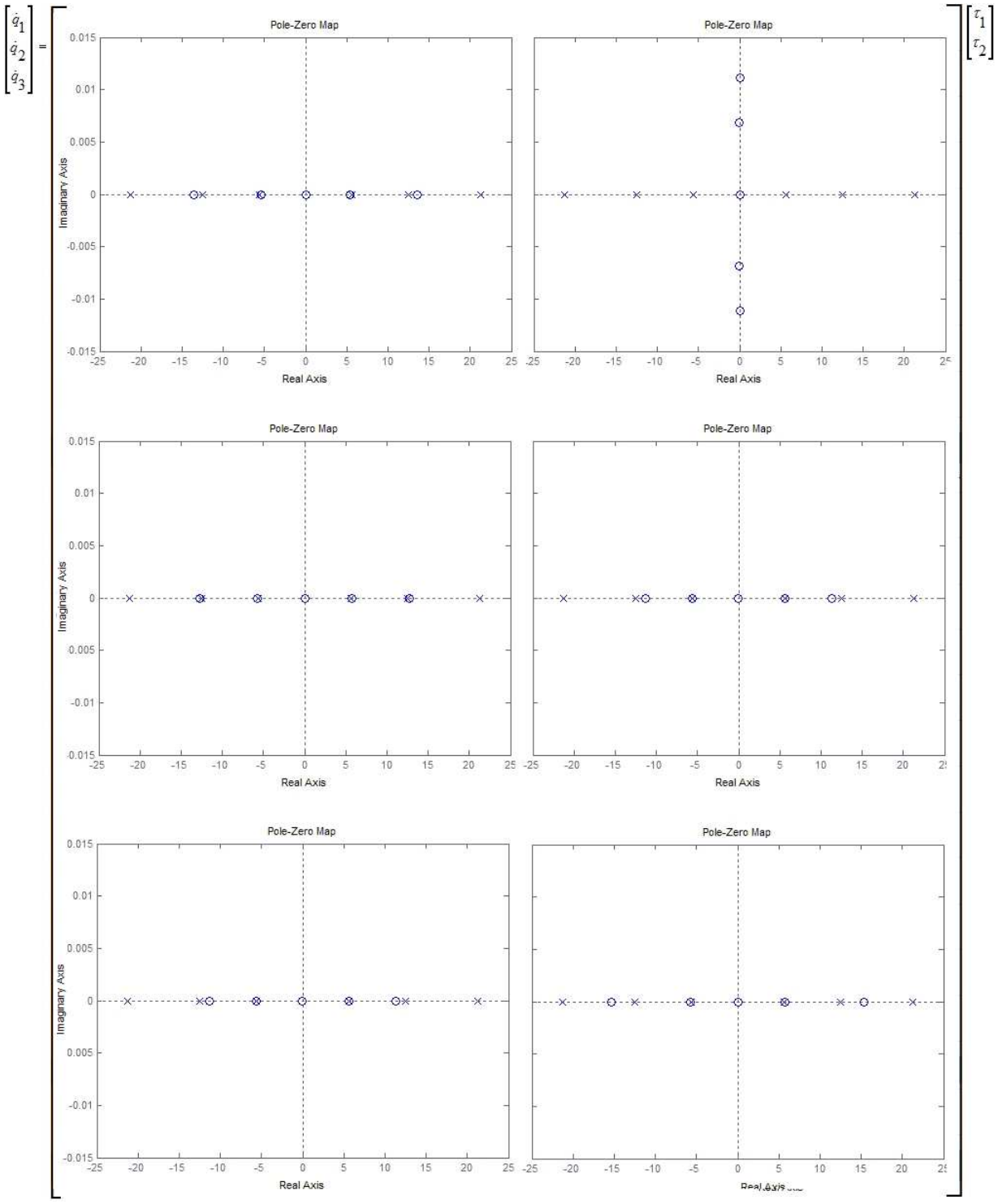


Figure E.2: Poles and zeros for the last 3 states ($\dot{q}_1, \dot{q}_2, \dot{q}_3$)