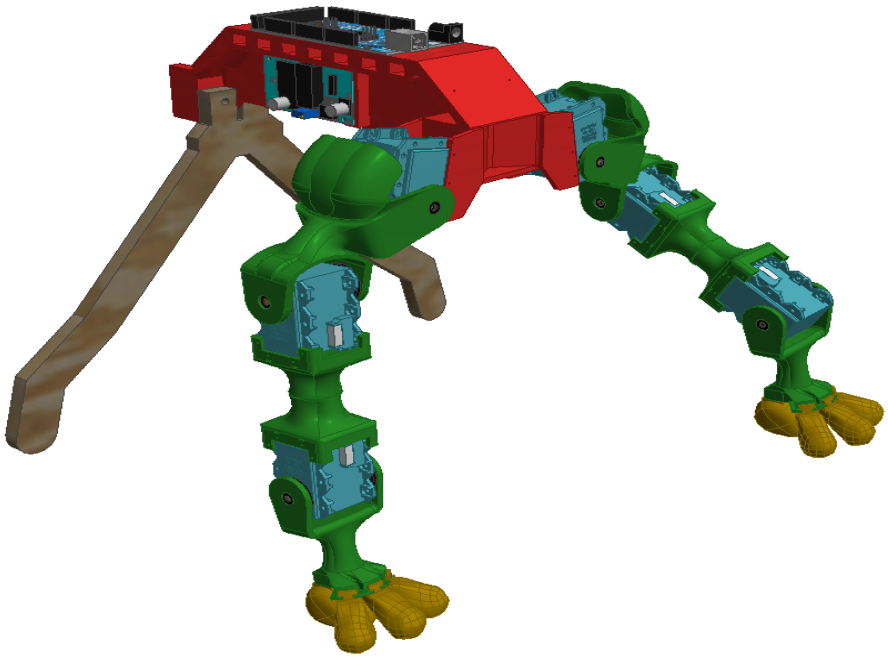
DEGREE PROJECT, IN MECHATRONICS , FIRST LEVEL
STOCKHOLM, *SWEDEN 2015*

The Paralyzed Quadruped

A COMPARISON OF TWO GAITS

HENRIK LJUNGGREN



KTH ROYAL INSTITUTE OF TECHNOLOGY

ITM

TRITA MMK 2015:14 MDAB067

KTH Industrial Engineering and Management

Bachelor's Thesis MMKB 2015:14 MDAB 067

The Paralyzed Quadruped

Henrik Ljunggren

Approved	Examiner	Supervisor
2015-05-13	Martin Edin Grimheden	Didem Gürdür

ABSTRACT

In this report the significance of the number of degrees of freedom for walking robots is evaluated. To test this, a robot of two functioning legs with 3 degrees of freedom for each leg and two stiff support legs is built. This robot is then capable of walking with one of the two algorithms programed; one algorithm corresponds to one gait. The first gait utilizes 3DoF to walk and turn and the second gait uses only 2DoF while having one of the joints locked. The robot is then tested by measuring the time it takes to walk a set distance for three different scenarios. These scenarios are walking downhill, walking uphill and walking on a flat surface all without turning. The 2DoF gait resulted in being slower and having higher energy consumption but the data was inconclusive and more tests should have been made to further come to a complete conclusion.

KTH Industriell teknik och management

Kandidatarbete MMKB 2015:14 MDAB 067

Den Paralyserade Fyrfotade Roboten

Henrik Ljunggren

Godkänt	Examinator	Handledare
2015-05-13	Martin Edin Grimheden	Didem Gürdür

SAMMANFATTNING

I denna rapport disskuteras vikten av antalet frihetsgrader för en gående. För att testa detta byggdes en robot med tre frihetsgrader per ben. Roboten kan gå med en av de två algoritmer som programerats och där varje algoritm motsvarar en gångstil. Den första gångstilen använder sig av alla de tre frihetsgrader per ben för att gå och svänga och den andra gångstilen använder enbart två frihetsgrader genom att låsa en av lederna. Robotens två gångstilar är sedan testade genoam att mäta den tid det tar att gå ett fixt avstånd i tre olika förhållanden. Dessa är gå upp för en backe, ner för en backe och på plant golv. Gångstilen som bara använder två frihetsgrader slutade med att vara långsammare och ha högre energikonsumtion. Resultaten tagna från testerna visade sig vara icke övertygande för att komma fram till en bra slutsats.

PREFACE

This section conveys gratitude to all people involved who have helped with the progress of this project.

I want to thank my supervisor, Didem and the assistants of the mechatronics lab for giving me advice during the course of the project. I also want to express my gratitude to Per Von Wowern for the much appreciated help with 3D printing the structural parts.

Henrik Ljunggren

Stockholm, Maj, 2015

CONTENTS

NOMENCLATURE

This section introduces the symbols and abbreviations used in the following report.

Symbols

Symbol	*Description*
x	The x coordinate of the foot [mm]
y	The y coordinate of the foot [mm]
z	The z coordinate of the foot [mm]
v_1	Angle for the topmost joint
v_2	Angle for the middle joint
v_3	Angle for the lowest joint
L_y	Length of shoulder in the y direction
L_z	Length of shoulder in the z direction
L_2	Length thigh
L_3	Length shin
$\overline{x}$	The x components of the trajectory
$\overline{y}$	The y components of the trajectory
$\overline{z}$	The z components of the trajectory
k	The proportional constant
dir	The direction when turning
l	Indicates if left or right leg
$\overline{T}$	The turning offset vector

Abbreviations

CAD	Computer Aided Design
RAM	Random Access Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
ID	Identity
DoF	Degrees of Freedom
UART	Universal asynchronous receiver/transmitter

1 INTRODUCTION

This section will describe the project background, purpose and scope as well as the methods used in order to reach the set goal.

1.1 Background

Legged robotics has become a fast growing technology and the reason being a demand in more adaptive locomotion in new fields of applications. Wheeled locomotion is in general more energy efficient than legs on flat surfaces with good traction but when it comes to rough terrain legged locomotion takes over [1].To be able to navigate through rough terrain the capability of choosing a sound foothold is essential in order to traverse forward. With the help of evolution animals have developed efficient and precise legged locomotion something today's technology has yet to achieve [2]. But eventually, as movement capabilities of legged robot advance, so will their applications in various fields. The new legged robots could help in rescue operations, unmanned exploration, pack-mules, and military operations [3].

This project will try to shed light on the difference of using 3DoF legs compared to using 2DoF legs. Intuitively a higher mobility leg such as a 3DoF leg will be better at walking and turning with its improved movability however it will also require a more complex and expensive system. It is therefore of great interest to evaluate if a robot's task requires the extra movability or if a simpler model will be sufficient. This is something that will be examined in this report.

1.2 Purpose

The purpose of this paper and project is to determine the significance of the knee joint in walking robots. A human leg has for example 6DoF per leg, 3DoF for the hip, 1DoF for the knee and 2DoF for the ankle [4]. The robot of this project will have at most 3DoF per leg, a 2DoF hip and a 1DoF knee. But to test the purpose of this report the knee joint will be locked, which can be described as to walking on a pair of stilts or by having your knee joint *paralyzed*, walking is still possible but only to a certain extent. This has led to the following research question.

> *How will the walking speed and power consumption be affected with using 3DoF compared to only 2DoF for each leg?*

To examine this, several tests will be performed on a robot which will be built as a platform for this project and by answering this research question, the construction of legged robots and the consequences of choosing a lower number of joints will be better understood.

1.3 Scope

Only one type of quadruped will be examined. The quadruped will have 3 DoF per leg and have a horse inspired design. In order to make programing the gait of the robot easier and less time consuming only the two front legs will be used when examining the research question. The two rear legs will be replaced with two completely stiff legs which will be dragged along. The robot will also only be walking and tested on flat

surfaces, never on rough terrain. This way the mobility of a 3DoF could prove to be redundant instead of vital for a rough surface. One step for each gait will be programed to take the same amount of time, this is done to be able to compare later on.

1.4 Method

To move the robot in a controlled and synchronized manner each leg is considered to be separate from the others with its own local coordinate system. The local coordinate systems have their origin at the topmost servo's axle, see Figure 1. Having the coordinate systems not connected introduces a complication at the cost of making the motion of the leg easier. The complication is the uncertainty of the actual position of the origin compared to the ground. To get around this the gait is programmed so that the main body will keep at a constant height. This way the end effector, the heel of the foot will correctly follow the predicted trajectory.

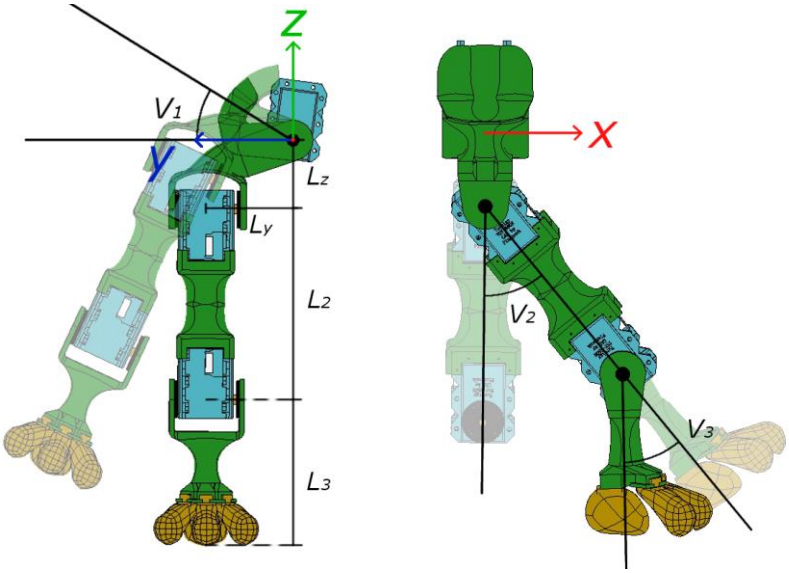


Figure 1 Cartesian coordinates and servo angles definition

The basic trajectory of one leg will have the end effector following a second order equation. However a leg can only move within its space of influence and certain positions are therefore not valid. To test which trajectories were acceptable a simulation done in Matlab was programmed. In Matlab a leg was described by cylinders in a 3D plot moving with time as the limbs of the robot would, see Figure 2. Though the actual dynamics of the limbs are overlooked the simulation gave a clear picture of what to expect of the robot making debugging easier. The actual trajectory for the physical robot could conversely vary from the simulated trajectory done in Matlab.

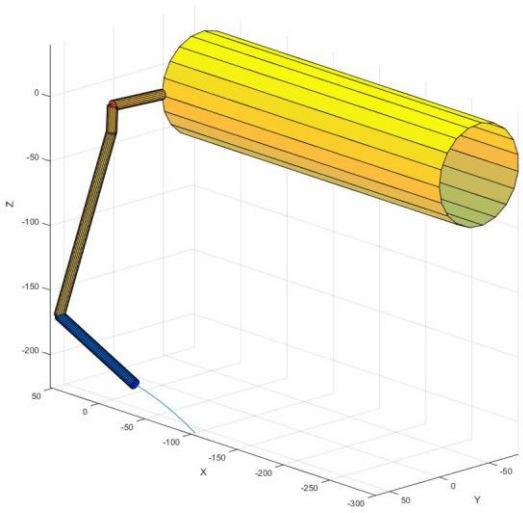


Figure 2 Simulation done in Matlab

To actually follow the set trajectory two distinct ways are either program the servos to go to the next point in the trajectory or only regulate using the angular speeds. Using the latter ensures a smoother motion and will make changing the walking speed easier but all servos need to be perfectly in synchronization with each other and constantly change their speed according to a function given and calculated by the microcontroller. To correctly follow the trajectory, the movement of the servos will have to be derived from a closed loop system or else an increasing error with time will occur. Without feedback the legs' positions will drift away from the correct trajectory. The easiest method is to use a proportional controller for each servo with the current angle as input and the angle difference as feedback for calculating the new angular speed. This controller will therefore adjust the speed accordingly to the difference in current and the ideal position. To ensure synchronization between the legs the trajectory will consist of many points and with each point having a time stamp of when the leg should be located at that certain point.

A leg's trajactory consists of two states, the transport state for when the leg is in the air and the support state for when the leg is supporting the weight of the robot while translating the main body forward. Observing any creature walk makes it clear that in the transporting state the foot needs to be lifted from the ground and carried forward in a arc-shaped trajectory. Then later in the supporting state the foot is draged bakward and pulling the rest of the body forward. To stay balanced the feet will be close together much like a cat, the supporting leg will then be closer to the center of mass. In order to be able to control the motion in these states, the foot's location needs to be calculated and to do this forward kinematics is used. Forward kinematics implies transforming the servos' angles to the corresponding rectangular coordinates x, y and z [5]. However what's equaly important is going from the cartisien coordinate system to the corresponding angles for the servos. This is done with inverse kinematics, a transformation from the carstisian coordinate system to the coorisponding angles for the servos[5].

In order for the robot to turn, the trajectory for each leg will have to change. This implies that a new set of targeting points need to be calculated and that the timing of the legs needs to be modified to fit the transition from going straight to turning. When going in a straight line the foot leaves the ground at the same *y*-coordinate as it hits the ground after lifting the leg, see Figure 3. But when turning, the *y*-coordinate will shift and create an offset so that in the supporting state the front of the robot will drag itself sideways resulting in a turning motion.

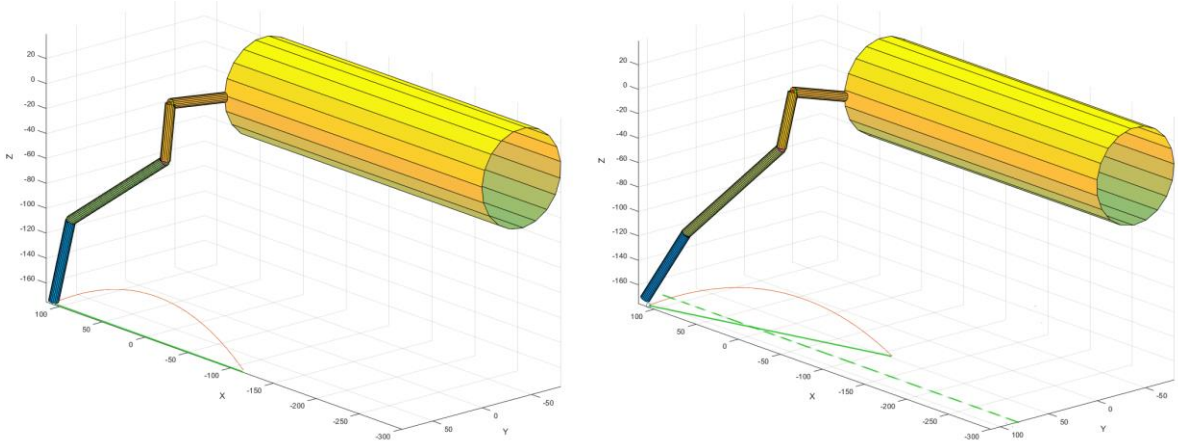


Figure 3 Turning motion in Matlab

When locking the knee joint the leg becomes only 2DoF and now an arbitrary trajectory can no longer be followed using only two servos. This means that the previous method of controlling the motion no longer applies for when locking one joint. This problem is worked around by choosing the trajectory carefully so that even though the same 3DoF algorithm is used only two servos will need to be regulated in order to follow the set trajectory. This is done by calculating the trajectory's *y* values depending on the *x* and *z* values so that the knee joint always is straight.

Turning with only 2DoF is done again by offsetting the start *y* from the end *y* coordinates but because the *y* coordinate cannot be set arbitrary anymore the start *x* and end *x* coordinates are instead shifted. A shift in *x* results in a shift in *y* which makes the robot turn. This in detail is later described in the theory section.

To be able to answer the research question, the walking speed of the robot and the power consumption will be measured. To measure the walking speed the robot is timed walking from start to goal at a set distance. The power consumption will be measured by repetitively checking the servos internal 10 bit load value with 1023 meaning the servo is applying full torque and zero being no torque is applied. From this data, conclusions about how the two designs are suitable for different tasks will be drawn.

2 THEORY

Here theory will be introduced. Knowledge gathered and needed to complete this project will be explained before used for a better understanding of the project.

2.1 Forward and inverse kinematics

The trajectories for the transporting and supporting states are described as a set of points, see equation(1.1).

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & z_n \end{bmatrix} \tag{1.1}$$

These points could be derived from any particular function but could also be hardcoded points depending on how the leg would move. Every position in the Cartesian coordinate system corresponds to either several sets of angles for the servos or no solution. This is because of how the leg is shaped and how the kinematic equations generally have multiple solutions [6]. This can be clearly seen in Figure 4; here two solutions lead to the same point. For one of the robot's legs two different solutions per point will be found. In this project and in the coding of the robot, the solution with the knee pointing forward is always chosen.

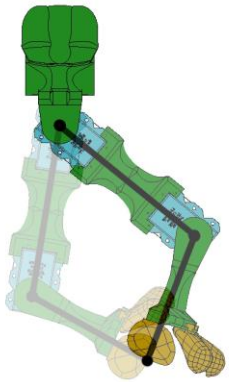


Figure 4 Inverse kinematics multiple solutions

The three angles v_1, v_2, v_3 are defined as positive as shown in Figure 1. When transforming from representing a position with servo angles to Cartesian coordinates forward kinematics is used. The forward kinematics' equations were calculated by simply adding up the translation of each section of the leg as

$$\begin{aligned} x &= L_2 \cdot \sin(v_2) + L_3 \cdot \sin(v_2 - v_3) \\ y &= L_y \cdot \cos(v_1) + \sin(v_1) \cdot (L_z + L_2 \cdot \cos(v_2) + L_3 \cdot \cos(v_2 - v_3)) \\ z &= L_y \cdot \sin(v_1) - \cos(v_1) \cdot (L_z + L_2 \cdot \cos(v_2) + L_3 \cdot \cos(v_2 - v_3)) \end{aligned} \quad . \tag{1.2}$$

Where L_y, L_z, L_2 and L_3 are the lengths as defined in Figure 1. For every set of angles only one solution of *x*, *y*, *z* is given. Transforming from a given Cartesian coordinate to a set of servo angles is however not as easy. Finding the inverse kinematics transformation can

generally be done in two ways, using inverse matrix of the forward kinematics or using a geometry approach [5]. In this project the geometry approach was used because of the relative simple model of at most 3DoF and to avoid a potential numerical calculation of finding the inverse matrix which could end up using valuable time for the microcontroller. To find the inverse kinematics' equations, the angle v_1 will be the first to compute given that it only has one solution regardless of the target Cartesian point. This is because orientation of the servo, the direction of the axle is not a duplicate like the other two servos are. The angle v_1 has to be set so that the leg will be pointing at the target coordinate in the *yz*-plane, see Figure 5.

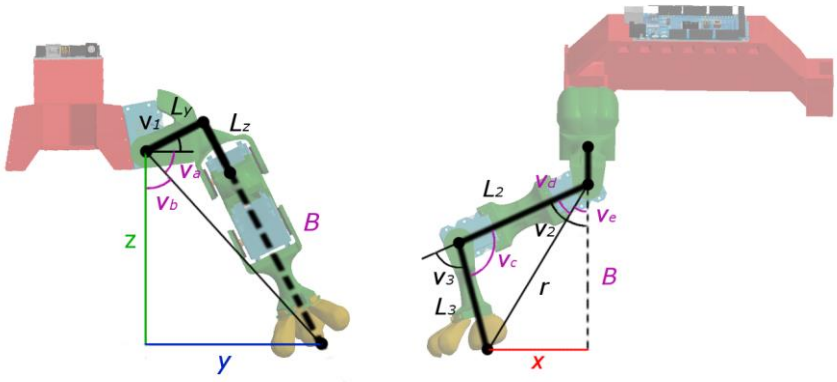


Figure 5 Inverse kinematics calculations

The angle v_1 is not depended on the value of x. The geometry in Figure 5 then gives the following solution,

$$v_a = \arccos\left(\frac{L_y}{\sqrt{z^2+y^2}}\right), \qquad v_b = \arctan\left(\frac{y}{-z}\right)$$
$$v_1 = v_a + v_b - \frac{\pi}{2} \tag{1.3}$$

When calculating the angles v2 and v3 the middle point of the axle of v2 and the distance r is needed. These are expressed as

$$y_1 = L_y \cdot \cos(v_1) + L_z \cdot \sin(v_1), \qquad z_1 = L_y \cdot \sin(v_1) - L_z \cdot \cos(v_1)$$
$$r = \sqrt{(0-x)^2 + (y_1-y)^2 + (z_1-z)^2} \tag{1.4}$$

The next angle to compute will now be v_2 with the use of the cosine rule and some basic trigonometry. This is where the solution of the knee pointing forward is chosen by making v_d positive instead negative. The calculations are then computed as

$$B = \frac{-(z-z_1)}{\cos(v_1)}, \qquad v_e = \arctan\left(\frac{x}{B}\right), \qquad v_d = \arccos\left(\frac{r^2 + L_2^2 - L_3^2}{2 \cdot r \cdot L_2}\right)$$
$$v_2 = v_e + v_d \tag{1.5}$$

To compute the final angle v_3 the cosine rule is used again but this time to find v_c. The sign of v_3 now has to correspond to the sign of v_d in order to stick with having the knee pointing forward. The equations then come to be

$$\begin{aligned} v_c &= \arccos\left(\frac{L_2^2 + L_3^2 - r^2}{2 \cdot L^2 \cdot L^3}\right) \\ v_3 &= \pi - v_c \end{aligned} \quad . \tag{1.6}$$

2.2 Motion of a leg

The value of $\bar{x}$ is set as points equally distributed from start to end value of *x*. The very motion of a 3DoF leg in the transporting phase is expressed using these vector equations with all compilation done element-wise,

$$\begin{aligned} & & \bar{x} &= \begin{bmatrix} -60 & \cdots & 60 \end{bmatrix} \\ \textbf{3DoF} & & \bar{y} &= -26 + \left(\frac{\bar{x}}{40}\right)^2 + \bar{T} \cdot k \cdot dir \cdot l, \qquad \bar{T} = \begin{bmatrix} -30 & \cdots & 30 \end{bmatrix} . \\ & & \bar{z} &= -185 - \left(\frac{\bar{x}}{8}\right)^2 \end{aligned} \tag{1.7}$$

Where $\bar{x}$ and $\bar{T}$ are a vectors with values linearly ranging from -60 to 60 and respectively from -30 to 30. These vectors and values are in millimetres and together model the transporting phase of one step. The $\bar{T}$ vector is used for turning and creates an offset in *y* in order to turn. The direction and magnitude of the turn is then determined by *dir* respectively *k*. To turn left *dir* is set to -1 and to turn right it is set to 1. The magnitude is a value ranging from 0 to 1 and decides how hard the turn will be. Left and right legs need to be shifted in opposite directions so by setting *l* to either -1 or 1 this function will work for all legs. A complete turning trajectory can be seen in Figure 3 for a 3DoF leg. The constant numbers in (1.7) and all the following trajectories are taken from the simulation in Matlab which gives a visual feedback if the motion is acceptable or not.

During the supporting phase the *x* and *y* components are calculated so that a straight line is formed between the end coordinate and the start coordinate of the transporting phase. This means that the *z* component will be constant if the step starts at the same height as it ends and that the *y* component will be constant if walking straight.

The motion of a 2DoF leg in the transporting phase is harder to expressed using equations especially when involving turning. Because of the limitations of only having 2DoF for any point in space only two coordinates can be selected whilst the last coordinate will be set by the geometry. The foot, the end effector will only be able to reach points on the surface on a sphere. For this robot *y* will be the geometry set variable. The transporting phase of a 2DoF leg can be seen in Figure 6 to the left and the supporting phase can be seen to the right.

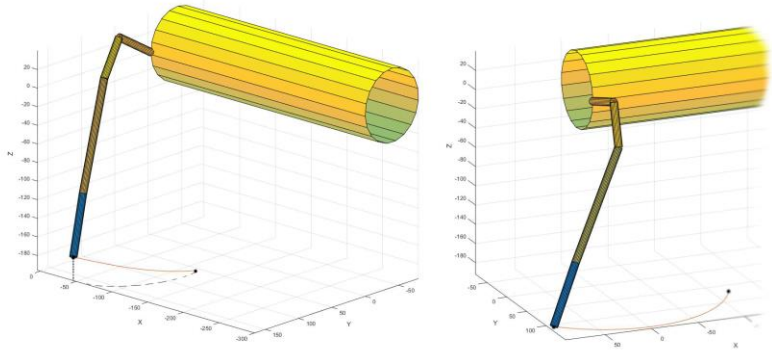


Figure 6 2DoF leg trajectory

When the *y* components no longer are selectable the *x* components will be shifted instead to achieve a turning motion. The start and end value of *x* are then calculated as

$$\textbf{2DoF} \qquad \begin{aligned} x_{start} &= -60 - k \cdot \left(5 \cdot \frac{1 - dir \cdot l}{2} - 20 \cdot \frac{1 + dir \cdot l}{2} \right) \\ x_{end} &= 60 + k \cdot \left(5 \cdot \frac{1 + dir \cdot l}{2} - 20 \cdot \frac{1 - dir \cdot l}{2} \right) \end{aligned} . \tag{1.8}$$

When for example turning right the x_{start} will be set to -90 and x_{end} to 115. For this trajectory *x* and *z* have been set by a certain function chosen to fit the purpose but given that the leg only has 2DoF *y* has to be strictly set by the geometry and based on the value *x* and *z*. The final functions are then expressed as

$$\textbf{2DoF} \qquad \begin{aligned} \bar{x} &= \left[x_{start} \quad \cdots \quad x_{end} \right] \\ \bar{y} &= \sqrt{L_y^2 + \left(\sqrt{(\mathrm{L}_2 + \mathrm{L}_3)^2 - \bar{x}^2} + L_z \right)^2 - \bar{z}^2} \\ \bar{z} &= -218 - \left(\frac{\bar{x}}{10} \right)^2 + dir \cdot k \cdot \frac{25}{400} \cdot \bar{x} \end{aligned} . \tag{1.9}$$

Unlike the *z* component in 3DoF a correction term for *z* is needed in order to centre it on the new mean value of *x* during turning when *x* is shifted.

To express the motion during the supporting phase a simple line would be impossible to follow for the 2DoF leg. Instead to ensure a constant value of *z* an arc-shaped trajectory based on the lengths of the limbs would have to be computed. The value of *y* then has to change depending on *x* and *z* same as before. Should the math be correct in the transporting phase, the end and start value of *z* will then be the same and will be set as the new constant value of *z* in the supporting phase. To make balancing easier the arc in the supporting phase is pointing inwards to the middle of the robot. The trajectory then becomes

$$\textbf{2DoF} \qquad \begin{aligned} \bar{x} &= \left[x_{end} \quad \cdots \quad x_{start} \right] \\ \bar{y} &= -\sqrt{L_y^2 + \left(\sqrt{(\mathrm{L}_2 + \mathrm{L}_3)^2 - \bar{x}^2} + L_z \right)^2 - \bar{z}^2} \\ \bar{z} &= z_{const} \end{aligned} . \tag{1.10}$$

2.3 Synchronized motion

For this robot only two legs will be used but the legs will still have to be synchronized with each other for the gait to work successfully. To achieve this, the angular speeds for the servos are set based on the positioning feedback from the servos. If a servo lags behind it will be set to a higher speed based on the current angle and the target angle. The microcontroller will update the speeds several times before switching to the next target point which will ensure a smoother motion.

When walking, a leg motion will follow a cyclic pattern and by observing slow-motion videos of animals walking at different speeds it's clear that this cyclic pattern generally stays the same regardless of the speed. The only thing changing is the speed of the motion and the phase shift between the legs, the gait. This means that only one motion needs to be programmed when going forward at different speeds. The gait is most easily explained looking at a horse; a horse uses amongst others the gaits walk, trot, canter and gallop depending primarily on how fast the horse wants to go [7]. A legs motion does not significantly change between changes in the gait unless the horse is trying to turn. The difference between gaits instead lays in the phase shift of one leg compared to another. A full gait cycle lasting one time unit is when a leg returns to the same position as it started from. In a normal walking gait every leg would be shifted 90 degrees or 0.25 time units but with only two front legs this will be 180 degrees or 0.5 time units. This means that just as one leg is finishing its transporting phase the other one will begin its. Figure 7 describes this by showing the time for when the foot hits the ground and the relative time duration a hoof is on the ground supporting the body [8]. Walking will be examined in this project which will ensure a stable static balanced gait. When using faster gaits the movement starts becoming dynamic which is harder to control.

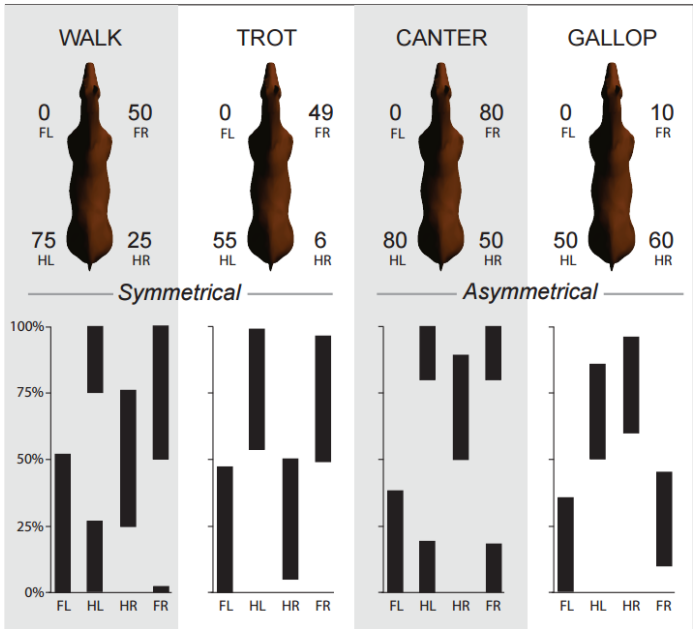


Figure 7 A horse's gait [8]

3 DEMONSTRATOR

In this section the demonstrator is described as well as the course of the project and the ending results. Many choices were made throughout the project and major ones are explained here.

3.1 Problem formulation

The robot will have to walk using three or two servos per leg and the very first design problem encountered is where and how to place the servos. This is important considering the type of gait wanted. Next problem is what kind of software and hardware to use and why. The hardware should not limit the gait too much but also not make it too complex resulting in making the software harder to implement. The robot should also have to withstand the torque from the servos and not break after a fall.

3.2 Software

The simulation was programmed in Matlab but the actual program for the robot was programmed in C. Many modifications were needed whilst converting the code in Matlab into the code for the real robot. The most vital functions and overall structure is explained in the below flowchart, see Figure 8.

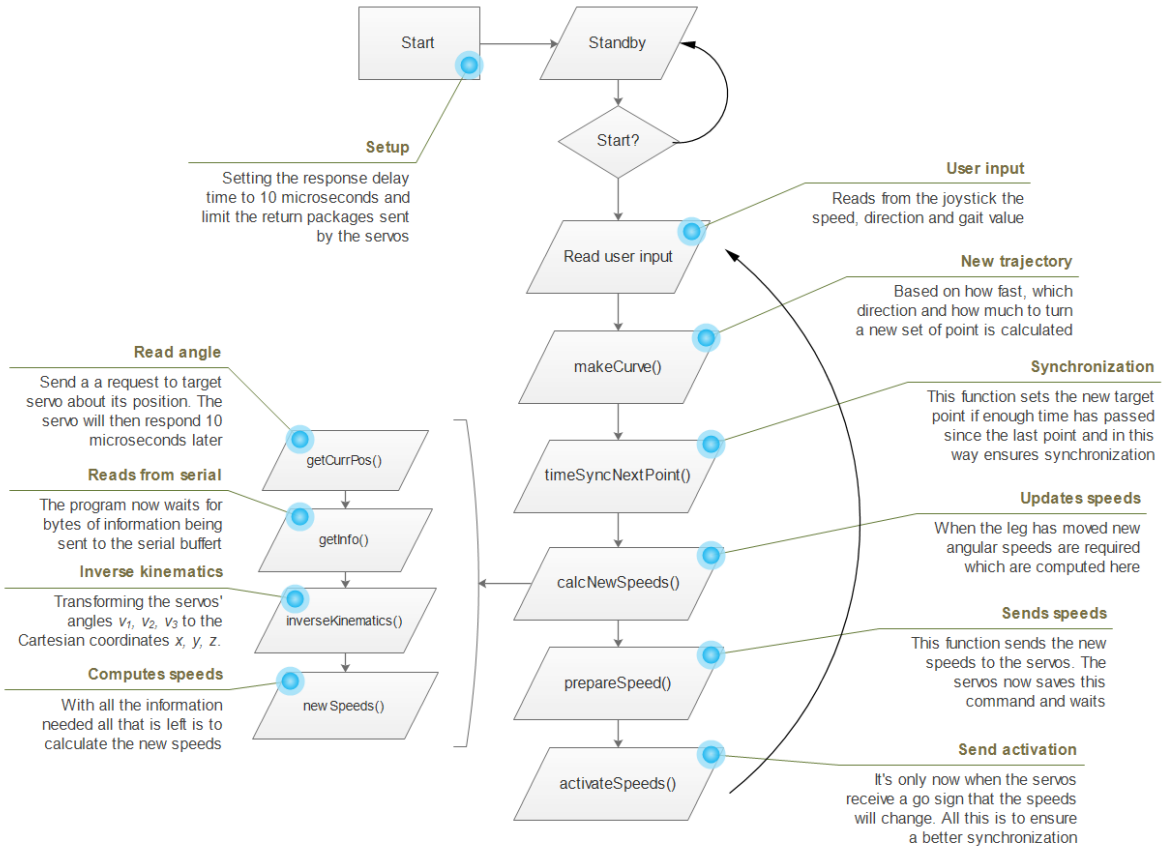


Figure 8 Program flowchart

The very first thing executed is setting the respond delay time for all servos. The respond delay time for a servo is the time taken from message received to when the servo sends back information. In order to have a fast and responsive system all time consuming functions will have to be set to a minimum and in this case 10 microseconds seems to be that limit, any faster and the received message becomes corrupt. In the setup the serial communication at a 500k baud rate is initialized. This communication speed can be changed but the current value is fast enough. At a 500k baud rate information is sent at a speed of 62500 bytes per seconds and the maximum serial speed the Arduino Mega 2560 can handle is 1M baud rate [9]. With a normal packet being about 8 bytes of data and about 10 packets are sent and received per leg the time actually spent sending and receiving information at 500k baud rate is about 1 millisecond which for this project is fast enough. Something which is also set during setup is limiting when a return package is expected from the servos. This is done to speed up the system by not having to wait for status packages when for example sending a packet for setting the angular speed.

The next step in the program is reading input from the user which involves reading the speed, direction, mode and value of *k.* The speed, direction and *k*-value are controlled by three potentiometers and read by using an analog digital converter on the Arduino. The 10 bit value from the analog input is then re-maped to either how many milliseconds between points, the percentage of turning and turning direction or the *k*-value for the proportional controller. After having this information the new trajectory can be produced and this is done for both legs separately for every time the program loops. To get best use of the Arduino and a smoother motion no delay is used to synchronize the legs with each other instead the program will update the speeds and trajectories as many times possible until enough time has passed so that a new target point is set. With faster updates a higher value of *k* is possible and therefore a lower static error.

To update the angular speeds of the servos information about the current location is needed. This information is requested from the servos one by one. After sending a packet containing a request for the current angular positon to a servo the program enters a reading serial information state. This state is exited with a time out if more than 1 millisecond passes but when the communication is working as normal this state is exited when 200 microseconds has passed since the last received bit of information. With all angular positions known the target angular positions are calculated by taking the Cartesian target coordinates *x, y, z* and transforming them by inverse kinematics to angular positions, see Appendix 3 for actual code. The angular difference will then be proportional to the angular speed for every servo. With a *k*-value of 1 the maximum speed will only be set to the servo if the angular difference is greater than $\pi/2$. When sending the new angular speeds to the servos, the change does not happen instantly instead then new command of changing the speed to the new speed is stored in the servos memory and only executed when the activation packet sent so all servos can read it is received. This activation packet is the last thing done before starting the loop over again. The reason is to ensure a better synchronization between the servos. The function used can be found in Appendix 4.

3.3 Electronics

The robot overall structure consists of a control unit, a power supply and six servos. For the control unit an Arduino Mega 2560 is used. The reason for not using a smaller microcontroller such as the Arduino Uno is being able to debug on the computer using the serial monitor. The Uno uses the same serial port for communication to the computer as it does for communicating with the servos. It is therefore better to use the Arduino Mega 2560 with its four serial ports so that one can be used to talking to the computer and the other talking to the servos.

The servos need a stable power supply and a certain voltage to function properly. To provide this a DC-DC step down converter is used together with a battery. The voltage can be set using the converter regardless if the voltage drops in the battery. This results in a stable adjustable voltage source. The speeds of the servos are highly depended on the voltage supplied meaning if the voltage would drop during testing the servos would slow down and affect the results.

A special circuit board which attach to the Arduino like an Arduino shield would was made to fit the communicating components, buttons for debugging, connectors for battery, servos, power switch and DC-DC step down. The circuit schematics can be found in Appendix 1.

There are two kinds of servos usable for this project, normal hobby servos or robotics servos. The difference is control, a normal hobby servo can be told to go to a particular position but the controller never knows when or if the servo actually arrived at the set position. This usually makes the algorithms wait for a set time roughly calculated depending on the angular distance. The angular speed will also not be adjustable and given all this, the movement will become choppy. But by using the robotics servos many additional tools are available such as speed and torque control, present and target position, temperature and many others. This is possible due to the small computer inside each and every servo. To communicate with the servos a packet of bytes is sent from the Arduino using UART which is in turn converted into the communication protocol RS485 using a small chip called SN75176BP, see Figure 9 and see the pin configuration in Appendix 2.

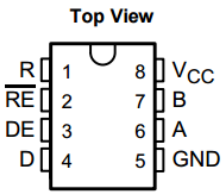


Figure 9 Communication curcuit SN75176BP [10]

All servos are connected as a daisy chain so all servos receive the same packages. To separate the servos from another every servo has an ID number and will only respond to the packet if it is marked with the corresponding ID number. The packages have a specific format see Figure 10 in order for the servos to interpret the information conveyed. A sent package always starts with the two bytes of [0xFF 0xFF] followed by one byte with the ID number of the receiving servo. Next is the length which is calculated as the number of parameters plus two. How many parameters sent depends

on what type of instruction was used. Next byte is the instruction byte, a byte telling the servos if the commands read or write will be used and following it are all the parameters, one per byte and finally at the end a not check sum to make sure the servo not received corrupt data. A returning package works almost the same as a sent one does but with the difference that instead of a byte with instructions a byte with what kind of error is returned. A sent package can either read or write information from or to the servo's internal memory. The servos memory consists of the types, RAM and EEPROM. RAM is set to default and EEPROM will be remembered after reboot of the servo. For example, sending a write package and saving a new speed value to a servo will change the set speed for the servo instantly. The address in with to save the new value is for speed control address number 32 and 33. Find the full control table at [11].

OXFF	0XFF	ID	LENGTH	INSTRUCTION	PARAMETER1	…	PARAMETER N	CHECK SUM

Figure 10 Sending Package

OXFF	0XFF	ID	LENGTH	ERROR	PARAMETER1	…	PARAMETER N	CHECK SUM

Figure 11 Receiving package

3.4 Hardware

Most of the structural parts were 3D printed using the Makerbot 5th generation. By doing this, constructing the complex forms of the different connectors wouldn't be a problem and it also helped during designing by having full scope. The plastic parts are also relative durable and if broken could easily be replaced and enhanced. The main body was also covered with a coat of epoxy to prevent the plastic layers from separating and reinforced with laser cut wooden composite at places where the servos were connected.

The design of the overall robot is an attempt to resemble a dog or any such similar creature. The servos are placed to best describe a dog's leg's motion based on the fact that in the animal kingdom heavy creatures, such as horses, elephants and dogs all have similar legs construction. This seems to be an appropriate design when supporting a lot of weight, meaning smaller motors with respect to the weight of the robot. The hip could be constructed in two ways depending which servo was placed topmost. The current design was chosen for the reason being when locking the knee joint this design gives a longer leg and therefore a longer reach. The size of the robot and foremost the length of the legs was roughly calculated with regards to the maximum motor torque so that the shoulder servo, the one with the lowest leverage would be able to lift a quarter of the robots weight when having a fully extended leg to the side.

At the bottom of the foot small pads are mounted. These pads are inspired from a dog's foot pads and will ensure a good contact area and extra friction when walking. The pads are also 3D printed but a rubber material was instead. The pads are also mostly hollow to increase deformation when pressing down the foot in the ground making it easier to walk on rough surfaces.

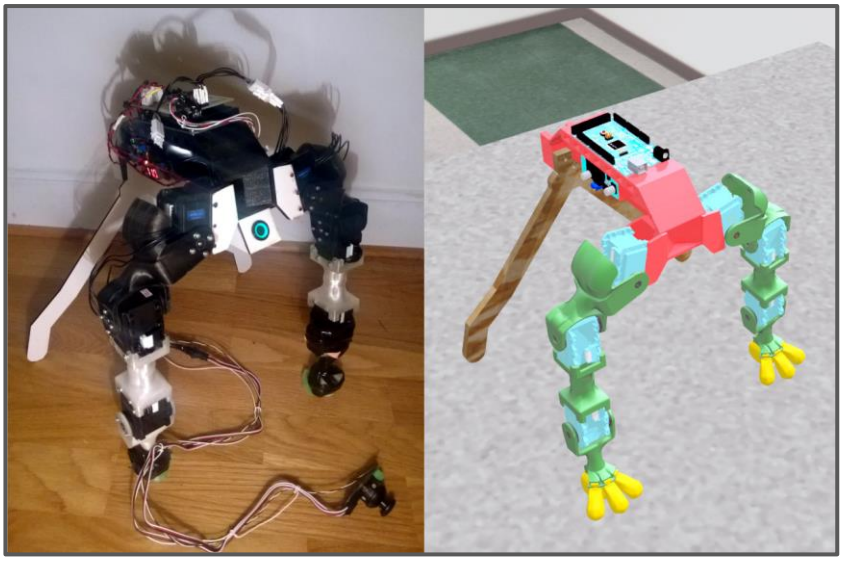


Figure 12 Left; actual robot, right; robot in CAD

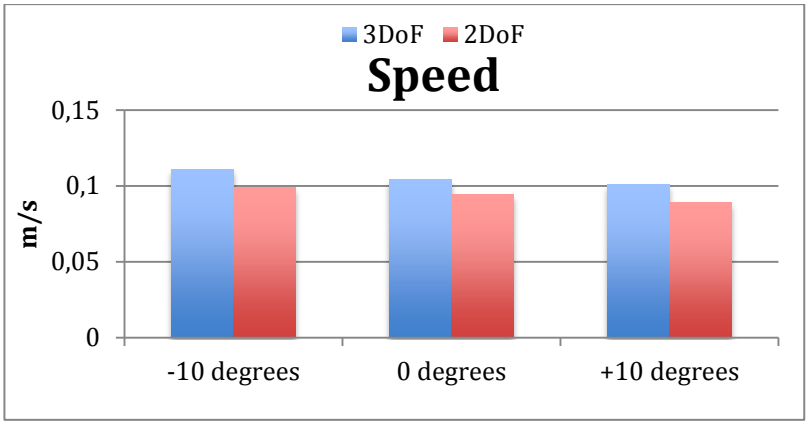
3.5 Results

Tests were performed to measure the time and power consumption of walking a set distance. 18 tests were performed in total, 9 done in 3DoF mode and 9 done in 2DoF mode. Three scenarios were tested, walking up and down a slope and on a flat surface. All tests were performed indoors with a fully charged battery. When testing walking up and down a slope a 1.5 meter long plank tilted 10 degrees was used. In Table 1 data of the performed test can be found.

Table 1 Test results

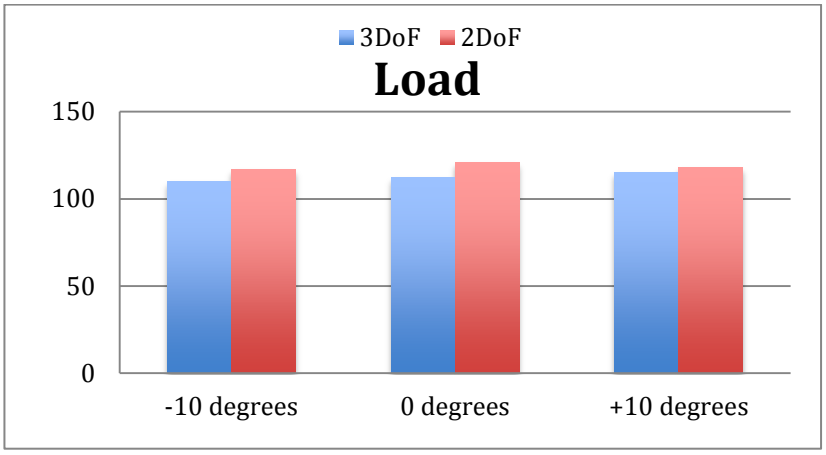
Test [*nr*]	DoF [*nr*]	Slope [*deg*]	Distance [*m*]	Time [*s*]	Speed [m/s]	Load
1	3	+10	1,5	15,0	0,100	120
2	3	+10	1,5	14,7	0,102	109
3	3	+10	1,5	14,9	0,101	115
4	3	0	2	19,0	0,105	105
5	3	0	2	20,3	0,099	119
6	3	0	2	18,6	0,108	112
7	3	-10	1,5	14,4	0,104	106
8	3	-10	1,5	13,1	0,115	124
9	3	-10	1,5	13,0	0,115	101
10	2	+10	1,5	15,5	0,097	115
11	2	+10	1,5	17,9	0,084	118
12	2	+10	1,5	17,4	0,086	121
13	2	0	2	22,1	0,090	120
14	2	0	2	22,2	0,090	124
15	2	0	2	19,6	0,102	118
16	2	-10	1,5	15,4	0,097	123
17	2	-10	1,5	14,7	0,102	110
18	2	-10	1,5	15,4	0,097	112

The results from Table 1 are presented in graph form in Graph 1 and in Graph 2. In Graph 1 the mean speed is shown for the three scenarios, walking downhill, on a flat surface and uphill. It can be noted that the speed of 2DoF is strictly lower than of 3DoF in all cases.



Graph 1 Showing the mean speed

In Graph 2 the combined load of the servos is presented for all three scenarios. It can be noted that the load for 2DoF does not change much depending on scenario.



Graph 2 Showing the mean load

4 DISCUSSION AND CONCLUSIONS

Here the results will be summarized and evaluated in order to try to answer the research question. Conclusions are then reached and presented.

4.1 Discussion

Even though the length of step and the duration of one full step are the same the 2DoF was slightly slower in all scenarios, this most likely has to do with the gait. When shifting from transporting phase to supporting phase in the 2DoF mode the leg swipes in from the side with very little height margin. Because of the small margin the foot touches the ground sooner than expected and this causes the length of steps to become shorter than it's supposed to be. In the 3DoF mode the leg is retracted by rotating the knee joint so that the margin to the ground increase fast compared to the swiping in from the side with the 2DoF gait.

Looking at Graph 2 a very small difference between the 2DoF and 3DoF mode can be noted. From a statistically point of view the difference would be too small to support any theory, one can only speculate at this point. One reason the 2DoF load is strictly higher than that of the 3DoF load could be how the 2DoF trajectory is shaped. The trajectory is much longer length wise but it is also curved even in the supporting phase. This means that when supporting the weight of the robot, the body is moved in an arc forward which when walking results in a rocking motion. What's happening now is that the servos need to accelerate the weight of the robot to the side only to accelerate it back again. This rocking motion comes at the cost of energy which could be one of the reasons the load for the 2DoF gait is higher.

The power consumption for the robot is mainly based on the required force and angled rotated by each servo. However power is also drained when countering a static force but without rotating the servo. This is the case in the 3DoF gait. When walking on a flat and even surface the topmost servo barely moves and mostly counters the static torque. This contributes to higher power consumption and because of the simple algorithm used to control the robot, not to stability. It is only when implementing a smarter walking algorithm, one which actually utilizes the extra servo for stability that the extra servo becomes a benefit for walking straight on flat surfaces. The robot will not always be walking forwards and should have to be able to turn. This is when the 3DoF design has a clear advantage over the 2DoF design and proves it's not redundant. With the help of the extra mobility the foot can be placed on a much larger area than without the extra servo. A controlled turning motion can then be executed.

It's worth noting that the speeds for the two gaits are almost the same which implies either one is good. Therefore by only looking at this as evidence the 3DoF design is redundant, a 2DoF would be about as fast and less complex and cheaper to build. This is of course true only if looking at this particular design and task. Constructing a robot often involves making it capable of handling more than just one task. If for example a robot was being built to transport material from one location to the other, the robot would then have to overcome all obstacles on the way to reach its target goal. Often if not always is this broken down into several problems which should all be solved by the robot without too much information. Being able to adapt is now crucial and being able only to walk on flat surfaces then has its limitations. But if the task at hand is well known

and simple a design can then be optimized for that particular task. The robot has no need to adapt and the extra movability then becomes unneeded.

An application for this type of research could be if a walking robot which normally uses 3DoF design with resemblance to this project is forced to swap to a 2DoF gait because of a damaged motor. The robot now has to evaluate if the its task still is accomplishable and to do this it first need to know its limitations such as walking speed and power consumption.

To fully compare the use of 3DoF legs and 2DoF legs more tests and variants of the design should be made. Only testing with one type of design limits what can be learnt from this project and applied to other robots. Examples of other designs using only 2DoF are hexapod and worm design. Both of which could prove to be much better suited to certain tasks. It is therefore hard if not impossible to prove whether a 3DoF leg design is redundant or a 2DoF leg design is not sufficient in all possible scenarios and tasks.

4.2 **Conclusions**

In conclusion, these tests are not enough to fully answer the research question. Even though a difference was measured the level of significance was too small to confirm with confidence the opinions introduced in the previous chapter. Nevertheless the results showed that with no real difference in speed and load having one extra degree of freedom proved redundant. However when comparing the stability or the movability of the two gaits, the 3DoF gait was far more superior to the 2DoF gait. Therefore if price and simplicity is important and the task is simple and well known a lower degree of freedom is to be preferred but if movability and the ability to adapt is more important, then a higher degree of freedom design would be more suited.

5 RECOMMENDATIONS AND FUTURE WORK

In this chapter future work and recommendations are offered for readers wanting to know more about the subject.

5.1 Recommendations

Something which could have been done better in this project is the stiff supporting legs at the back of the robot. These legs did not fulfill their use and should have been replaced by sturdier and heavier design with wheels. By having to hold the robot while performing tests in order to stabilize it involves a lot of error sources. Having a gantry on wheels would minimize sources of error.

5.2 Future work

The next natural step would be adding the two missing back legs. With the use of all four legs tougher terrain could be attempted and the significance of the required number of degrees of freedom could be accessed to a much higher degree. Also more dynamical gaits which could change the scope entirely should be tested.

Hardcoding the motion often has its limitations. In this case the stability of the robot does not take into account disturbances such the dynamical motion and changes in the surroundings. However, creating an algorithm so advanced which makes a robot able to go head to head with real animals in regards to stability and speed is extremely hard. Universities around the world are still trying to achieve the very basics of walking and running. To learn from the best, animals aren't programed to walk they are thought how to walk. All that is needed is the basic structure of the motor activity and a smart learning algorithm. A future work is to program such a algorithm and have the robot working for you instead the other way around.

REFERENCES

[1] Ignell, N.B., Rasmusson, N., Matsson, J., (2012), *An overview of legged and wheeled robotic locomotion*, Mälardalen: Mälardalen's University, pp 8.

[2] Carbone, G. & Ceccarelli, M., (2005) *Legged Robotic Systems*, accessible: http://cdn.intechopen.com/pdfs-wm/33.pdf, Rijeka: Intech [2015-05-12].

[3] Lee, K., (2014). *The Mathematical Model and Computer Simulation of a Quadruped Robot,* Milwaukee: Milwaukee School of Engineering, pp 1.

[4] Herman, I. P., (2007) *Physics if the Human Body*, New York: Columbia University, pp 6.

[5] Craig, J.J., (2005), *Introduction to Robotics*, Third edition, London: Pearson Education International, accessible: http://wcms.inf.ed.ac.uk/ipab/rss/lecture-notes-2014-2015/JJCraigRoboticsIntro.pdf , [2015-05-12], pp 112.

[6] O'Brien, J., (2005) *Forward and Inverse Kinematics*, accessible: http://www-inst.eecs.berkeley.edu/~cs184/fa05/lectures/lecture-19.pdf, [2015-05-12], Berkeley: University of California.

[7] Sutor, C., (1998) http://www.equusite.com/articles/basics/basicsGaits.shtml [2015-05-12]

[8] Murphy, J.E., Carr, H. & O'Neill, M., (2010) *Animating Horse Gaits and Transitions,* http://www.james-e-murphy.ie/files/murphy_animating_horse_gaits_2010.pdf, [2015-05-12], pp 2.

[9] Atmel corporation, (2011), *8-bit Atmel Microcontroller with 4/8/16K Bytes In-System Programmable Flash*, accessible: http://www.atmel.com/Images/doc2545.pdf, [2015-05-12], pp 197.

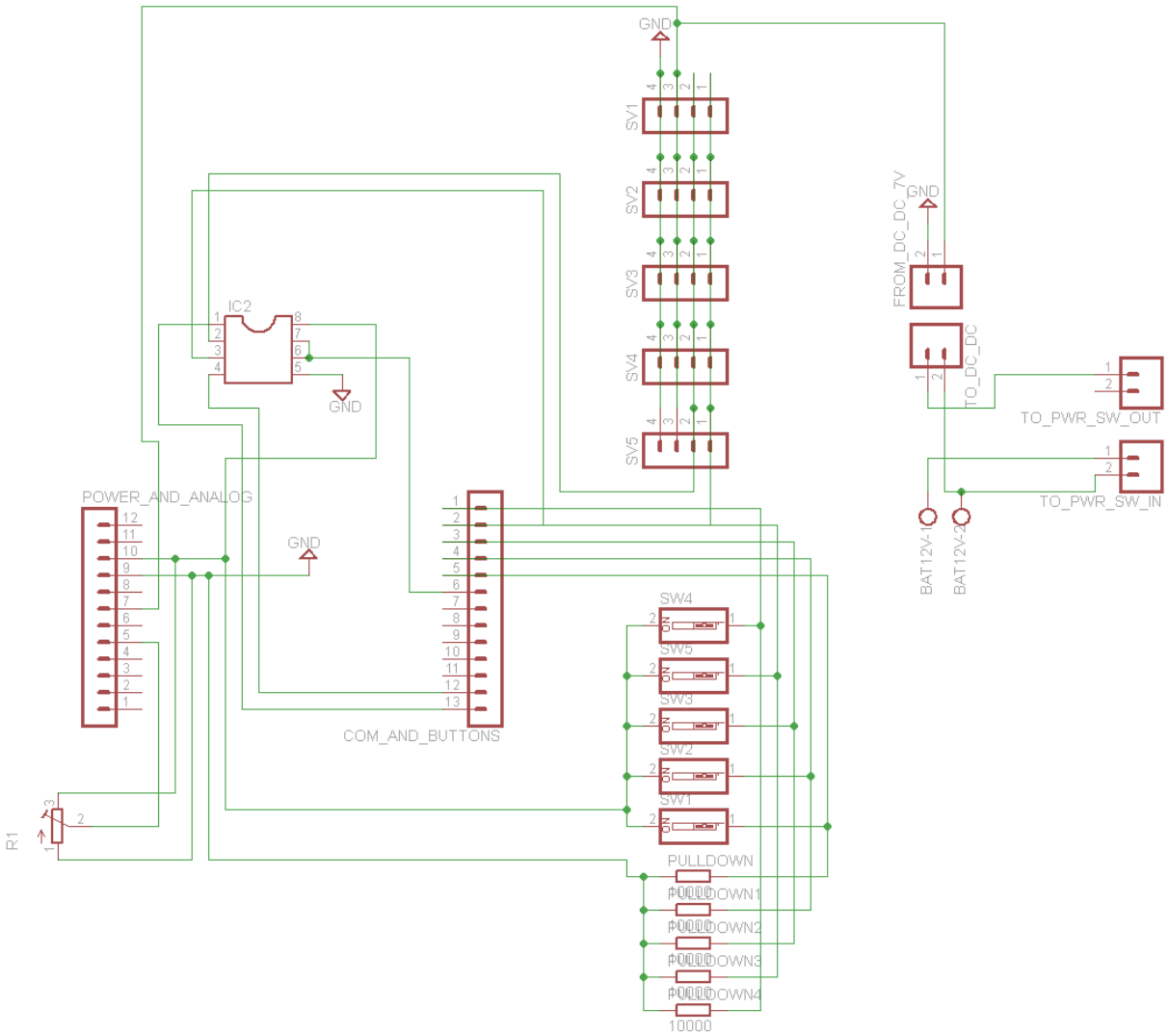
[10] Texas Instruments, (2003), *SN65176B, SN75176B DIFFERENTIAL BUS TRANSCEIVERS*, Texas: Dallas, accessible: http://pdf1.alldatasheet.com/datasheet-pdf/view/28198/TI/SN75176BP.html, [2015-05-12], pp 1.

[11] ROBOTIS CO., LTD, *User Manual Dynamixel RX 64,* Edition 1.1, http://creativemachines.cornell.edu/sites/default/files/RX-64_Manual.pdf, [2015-02-12], pp 21.

APPENDIX A: ADDITIONAL INFORMATION

This is where to find detailed material used and discussed in this project but which have no room or are not fit to have in the flow of the report.

Appendix 1 Curcuit schematics



Appendix 2 Pin Configuration SN75176BP

Pin Functions

PIN		TYPE	DESCRIPTION
NAME	NO.		
R	1	O	Logic Data Output from RS-485 Receiver
$\overline{RE}$	2	I	Receive Enable (active low)
DE	3	I	Driver Enable (active high)
D	4	I	Logic Data Input to RS-485 Driver
GND	5	—	Device Ground Pin
A	6	I/O	RS-422 or RS-485 Data Line
B	7	I/O	RS-422 or RS-485 Data Line
V_{CC}	8	—	Power Input. Connect to 5-V Power Source.

Appendix 3 Inverse Kinematics

```
void InverseKinematics(double x, double y, double z, float* angleCoords){

 float v1 = (atan(y/abs(z))+acos(Ly/sqrt(pow(z,2)+pow(y,2))))-PI/2;

 double x1 = 0;
 double y1 = Ly*cos(v1)+Lz*sin(v1);
 double z1 = Ly*sin(v1)-Lz*cos(v1);

 double r = sqrt(pow((x1-x),2)+pow((y1-y),2)+pow((z1-z),2));
 float v3 = PI-acos((pow(L2,2)+pow(L3,2)-pow(r,2))/(2*L2*L3));

 double B = abs(z-z1);
 float beta = atan(x/B/cos(v1));
 float v2 = acos((pow(r,2)+pow(L2,2)-pow(L3,2))/(2*r*L2))+beta;

 angleCoords[0]=v1;angleCoords[1]=v2;angleCoords[2]=v3;
 }
```

```
void calcNewSpeeds(int xtarget,int ytarget,int ztarget,int IDv1,int IDv2,int IDv3, int
Vspeeds[3], float* angleCoord, int leftOrRight){
 //leftOrRight [-1 | 1], -1=left, 1=right


 int v1Pos = getCurrPos(IDv1);if(corupptReturnData==1){return;}
 int v2Pos = getCurrPos(IDv2);if(corupptReturnData==1){return;}
 int v3Pos = getCurrPos(IDv3);//if(corupptReturnData==1){return;}

 float v1Curr = convertFromServoAngle(v1Pos)*leftOrRight;
 float v2Curr = convertFromServoAngle(v2Pos)*leftOrRight;
 float v3Curr = convertFromServoAngle(v3Pos)*leftOrRight;

 InverseKinematics(xtarget, ytarget, ztarget, angleCoord);
 float dv1 = angleCoord[0]-v1Curr;    //Still in Radians
 float dv2 = angleCoord[1]-v2Curr;
 float dv3 = angleCoord[2]-v3Curr;


 int v1Speed = map( dv1*1000,(-PI/2.0)*1000,(PI/2.0)*1000, -1023,
1023)*k*leftOrRight;    //Value can get higher than 1023 if the angle difference is bugger
than pi/4
 int v2Speed = map( dv2*1000,(-PI/2.0)*1000,(PI/2.0)*1000, -1023,
1023)*k*leftOrRight;    //Value can get higher than 1023 if the angle difference is bugger
than pi/4
 int v3Speed = map( dv3*1000,(-PI/2.0)*1000,(PI/2.0)*1000, -1023,
1023)*k*leftOrRight;    //Value can get higher than 1023 if the angle difference is bugger
than pi/4

 Vspeeds[0] = v1Speed;Vspeeds[1] = v2Speed;Vspeeds[2] = v3Speed;
 }
```