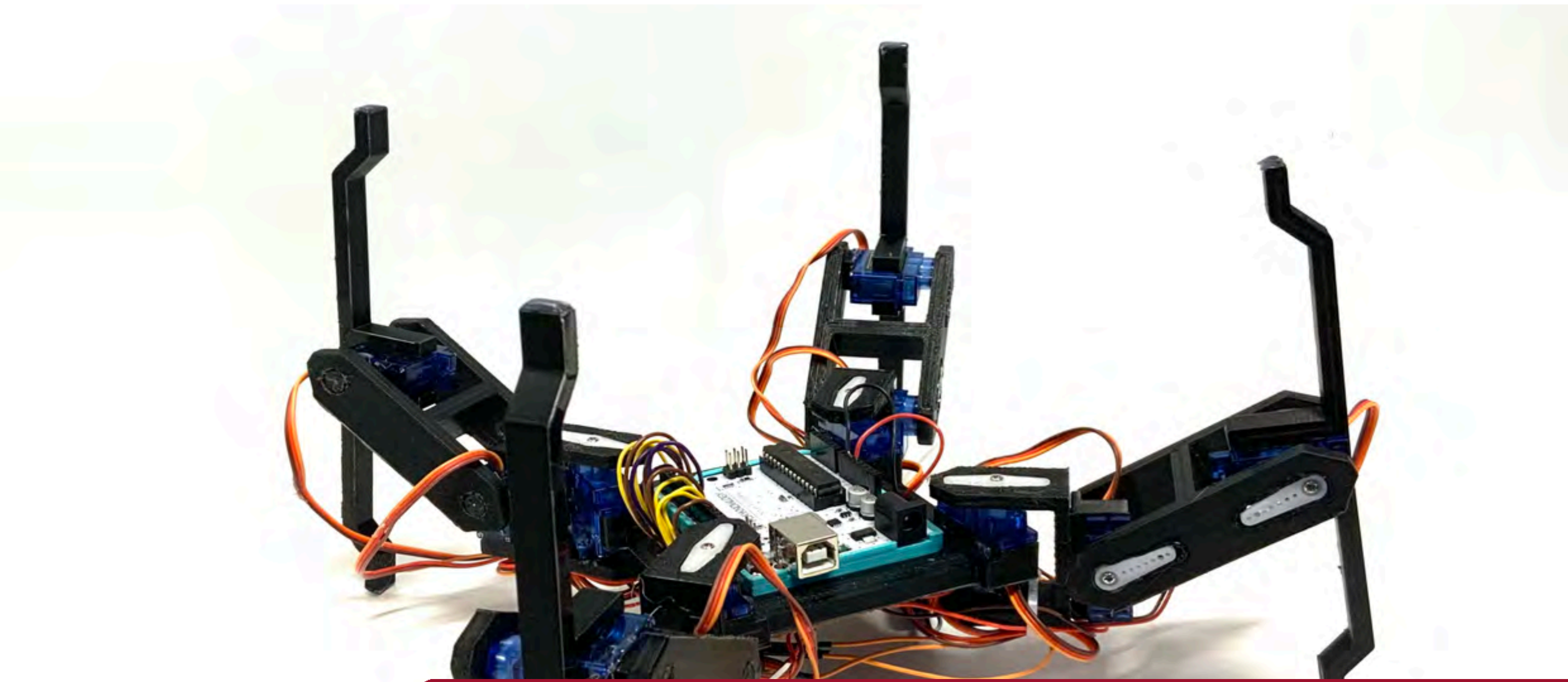
DEGREE PROJECT IN MECHANICAL ENGINEERING,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2021

Omnidirectional Quadruped Robot
Multidirektionell Fyrbent Robot

SAMUEL STENOW

SIMON LINDENFORS



KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT

Omnidirectional Quadruped Robot

SAMUEL STENOW
SIMON LINDENFORS

Bachelor's Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2021:35

Abstract

There are a lot of quadruped robots in the world, but few are omnidirectional. Therefore this thesis describes the production and design process of such a robot. Examining earlier quadruped robots determined that a central micro-controller is required to control it, and servo motors are used to power the robots joints. Reaserch also determined the base of the mathematical methods used. Additionally, there are multiple types of sprawling gaits, ranging from statically stable to dynamically stable. In this project a statically stable gait is used. The thesis illustrates the mathematical models used to define the omnidirectional movement, and describes the code used to implement it. The result is a robot that can move omnidirectionally, both normally and upside down. The results show that there is a deviation depending upon the direction, but it is small. The main advantage of omnidirectionallity is the ability to change movement direction without stopping or turning. It also enables directional adjustment without requiring any steps.

Keywords

Mechatronics, Robotics, Quadruped robot, Omnidirectional robot.

Referat

Multidirektionell Fyrbent Robot

Det här projektet gick ut på att skapa en krypande fyrbent robot som kan gå i alla riktningar utan att rotera runt sitt eget centrum. Det finns idag redan ett stort antal olika fyrbenta robotar, men få kan gå i alla riktningar. Därav så beskriver den här rapporten framtagningen och designprocessen för en sådan robot. Undersökning av fyrbenta robotar visade att en mikrokontroller är nödvändig för att kontrollera roboten och servomotorer bör användas för att driva lederna. Förstudeierna gav även basen för de matematiska modellerna som används för rörelserna, samt vetskapen om ett flertal olika typer av gångstilar, allt från statiskt stabil till dynamiskt stabil. I det här projektet beskrivs de matematiska modellerna som används för att definiera rörelsen i alla riktningar och hur dessa appliceras i programmeringen av roboten. Resultatet blev en robot som kan gå i alla riktningar utan att rotera runt sitt centrum, både normalt och uppochner. Detta ger möjligheten att byta rörelse riktning utan att behöva stanna eller vända sig, samt möjliggör även riktnings korrektioner utan att kräva extra steg.

Nyckelord

Mekatronik, Robotar, Fyrbenta robotar, Multidirektionell robot.

List of Abbreviations

PWM Pulse Width Modulation
DC Direct Current
RAM Random-Access Memory
EEPROM Electrical Erasable Programmable Read-Only Memory
PLA Polyactic acid
STL Standard Triangle Language
3D Three Dimensional

Acknowledgements

We would like to thank our supervisor Nihad Subasic and the Royal Institute of Technology for the opportunity, resources and help during the project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Throughout the world the industrial applications of robotics are large and there is no question that robotics are constantly being more and more implemented in our every day lives. Cars are on their way to become self driving, entire industrial facilities are able to run autonomously at night and the military are using a lot of funds to create the future of warfare. Many of the robots in use all over the world are mainly traveling on wheels, but are wheels the best answer? The majority of land is unpaved and not very accessible on wheels, for this environment walking robots are an important alternative thanks to their ability to move more freely and independent of flat surfaces [1]. There are many different ways to go on building a walking robot, there are biped robots, triped robots, quadruped robots, hexapod robots and so on. This thesis shows the construction of a quadruped robot with a sprawling-type of movement pattern resembling a spider [1]. Many of the quadruped robots you can find online are dependent on being positioned a specific way, but the authors of this thesis wanted to make a robot that was not dependent on this. In rough terrain where a multi legged robot is often made to travel, the terrain can possibly make the robot fall over. In order to eliminate this problem the robot in this thesis has the ability walk upside down without a problem.

1.2 Purpose

The purpose of this Bachelor's Thesis is to design and build a functioning prototype of a wireless walking four legged sprawling robot. The main research questions to be answered in this Thesis are,

- *How can a four legged robot be constructed to make it independent of what direction is up and down?*
- *How can a four legged robot be constructed to make it omnidirectional?*

- *What "gait" has an optimal stability to speed ratio?*

1.3 Scope

Due to this being a Bachelor's Thesis the time frame and funds were limited, constrains were necessary in order to make deadline and not go over budget. The main focus was to build the physical prototype of a four-legged robot and give it the ability to move in any direction, no matter which direction the robot was facing. As a secondary focus the robot was also built so that it could move independent of which side was currently upwards. If time allows testing will also be done to determine the optimal gait for the robot.

1.4 Method

In order to answer the research questions presented above different working methods were applied throughout the project. At an early stage research had to be done in the different areas and technologies used in the project such as microcontrollers, servomotors, movement algorithms and wireless connections. The information was collected from different scientific papers, articles and course literature.

Once sufficient theoretical information was obtained through research the design of the prototype was initiated. The main focus at this stage was to create the robot and its essential parts, the four legs. As a starting point a single leg was built with three servomotors controlled by an Arduino UNO. Once one leg functioned as desired, all four were built and assembled together on a main frame. The code was then developed so that the robot could move equally in all directions and upside down.

Chapter 2

Theory

The following chapter presents the necessary theory needed for the project.

2.1 Microcontrollers

Figure 2.1. Microcontroller Arduino UNO [2]

A microcontroller is a single Integrated Circuit used in many everyday appliances and tools, it gathers input, process information, and outputs a certain action. They operate usually at low speeds around 1MHz to 200MHz [3]. The main components of a microcontroller are an A/D converter, a microprocessor, Random-Access Memory (RAM), a flash memory, the Electrical Erasable Programmable Read-Only Memory (EEPROM), the Serial Bus Interface and the Input/Output ports. The microcontroller used in this project is an Arduino UNO, see figure 2.1, which has a lot of external components available at a low price [2].

2.2 DC servo motors

Figure 2.2. Servo motor [4]

A servo motor is a small and very energy efficient motor excellent for small or large project that require specific positioning of the shaft. Inside a micro servo motor used in this build, see above in figure 2.2, there is a small DC motor, potentiometer and a control circuit. As the motor rotates the potentiometers resistance changes, allowing the control circuit to regulate how much and in which direction movement is happening. A servo motor uses proportional control, meaning the speed of the motor is proportional to the difference between its actual position and desired position. The closer it is to its desired position, the slower it will move, allowing it to be very efficient. Servo motors are controlled with pulse width modulation (PWM), depending on the width of the pulses the motor will turn a specific amount. Once the shaft of a servo is in the desired position it will continue holding that position even if it is under external force, depending on how much force is applied in relation to the torque rating of the servo [5].

2.3 Movement of a four-legged robot

There are many different ways of moving a robot forward, so called ”gaits”. The purpose of this thesis to build a four legged robot, and for it to work well a good walking movement is required. The main different gait patterns are dynamic stable- and static stable gait. Dynamic stable moves the legs fast enough that the centre of mass does not have time to shift to an unstable position. Trotting gait is a dynamic pattern moving the diagonal legs simultaneously, this gait is fast but not very stable. Static stable moves one leg at a time, and shifts the center of mass away from the moving leg, making it very stable but slower than dynamic gaits. It is possible to combine the two methods by moving legs partly simultaneously and partly alone. This allows for some of the stability from static stable gait to be combined with some of the speed from dynamic stable gait [6]. An additional requirement to move the robot is that when each leg moves the robot has to shift its weight away from the leg so that the robot does not fall towards it when it lifts of the ground.

2.4 Inverse kinematics

A robots movement can be controlled in different ways, you can hard code the desired angles and speeds that motors need to move in order to get a certain movement for a arm or leg, or you can use mathematical expressions to calculate these angles depending on inputs. The second is called Inverse Kinematics and enables greater mobility, since it enables control of movement without calculating the appropriate angles for each step [7].

2.5 Solid Edge

To make a 3D model of the robot a software called Solid Edge was used. Solid Edge is a mechanical design system with many tools at hand to create 3D digital prototypes [8]. 3D modeling of a prototype before construction is extremely useful when figuring out the right design without spending time and materials to build something that wont work well in the end.

2.6 3D-Printing

In order to make the necessary parts for the robot that were not electrical components ordered online a 3D-printer was used. A 3D-printer takes a 3D model of the object to be printed in form of a STL file and makes it out of PLA plastic. The printing software also allows many different settings for the prints to reach the desired weight and detail. This method of constructing parts makes more complex and lightweight structures possible due to the precision of a 3D-printer and the low density of the plastic.

Chapter 3

Method

The following chapter describes the full process in detail for construction and testing of the robot.

3.1 Omnidirectional movement

Using Inverse Kinematics [7] to calculate the angles in the robots joints in reference to a predefined coordinate system enables the use of coordinates relative to each corner to place the legs. This in turn enables the use of other coordinate systems to control the positioning of the legs. This enables controlled movement that can be defined to create an omnidirectional area of available steps. Rotation matrices allow the calculation of the same movement relative to the body in each legs coordinate system.

3.2 Prototype construction

When planning the construction of the first prototype the main factors where to apply the concept of a sprawling-type robot, as seen in figure 3.1, as well as make the design symmetrical. Symmetry was required in joint placement, and the mobility of the joints. The end part of the leg also required symmetric design, to enable movement independent of upwards direction.

The first prototype made only consisted of three servo motors of model MS-1.3-9 [9] representing a single leg. From this prototype tests were made to see how well the servos worked together to move a leg and how a program for the microcontroller should be constructed. The next step included the making of a prototype in Solid Edge, see figure 3.2. Each leg has three parts, with a servo in each joint. The first part, closest to the center, enables horisontal rotation. The other two parts enable movement that keep a consistent height of the end of the leg, while also enabling the end to be further from or closer to the center. The part furthest from the center is also symmetrical to enable the robot to walk while upside down. When the design satisfied the needs for the robots movement, the parts were made with a 3D-printer

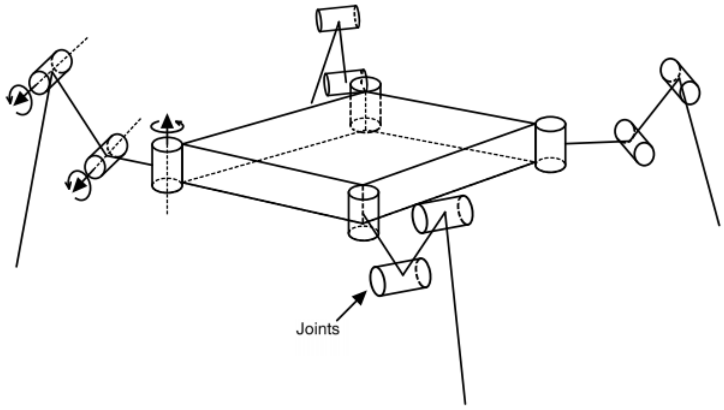


Figure 3.1. Schematic diagram of a sprawling-type robot, made by the authors.

Figure 3.2. 3D model prototype 2 designed in Solid Edge, made by the authors.

using PLA plastic. The settings for all of the prints were 20 percent infill and a printing speed of 40 mm/s. Once all parts were printed the second prototype could be assembled and the full software production for the testing of different walking movements could begin.

The tests on the first prototype revealed that though movement could be controlled correctly the size of the robot was slightly to much for the servo motors. To solve this a third and final prototype, shown in figure 3.3 was created. The third prototypes legs where constructed so that the horizontally moving part of the leg was closer to the center as well as shorter to minimize the momentum required from the servo motors. The end piece of the leg was shortened, though it remained longer than the middle piece. This enabled the robot to move its feet closer to its body, without changing its height. However, the middle piece was lengthened to maintain the possible range of the steps.

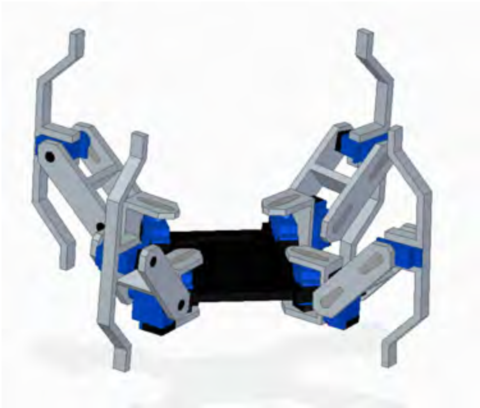


Figure 3.3. 3D model prototype 3 designed in Solid Edge, made by the authors.

The final prototype assembled with all components in place can be seen in figure 3.4.

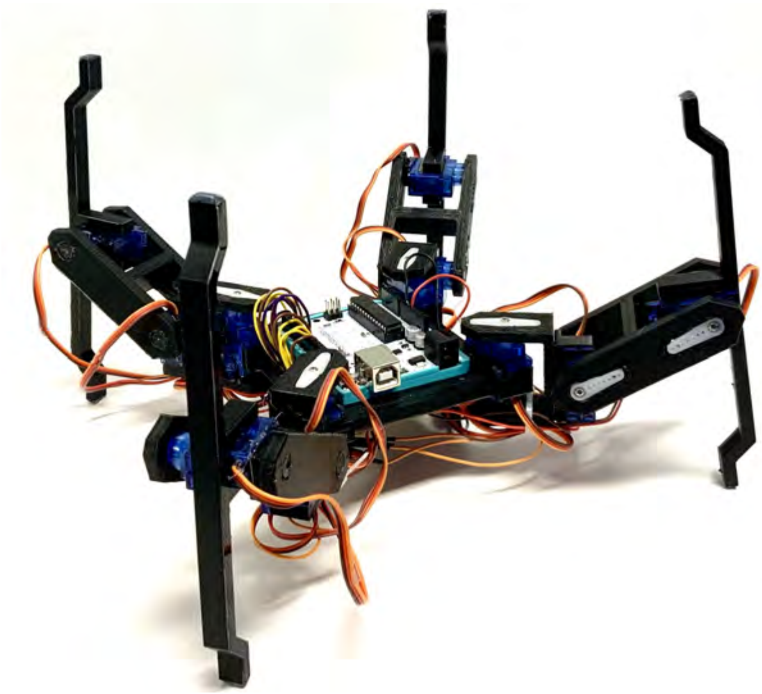


Figure 3.4. Final prototype, picture taken by the authors.

3.3 Software

The first program was built on the concept of static stable gait to get the robot moving in a stable but slow motion. The code was built in several functions, one calculated the angles the servo motors had to make relative to any end position of the robots foot during movement using inverse kinematics. The figures 3.5 and 3.6 below shows the trigonometrical problems that had to be solved in order to achieve this result. In order to calculate the desired angle θ_1 for the first joint, equation 3.1 was used.

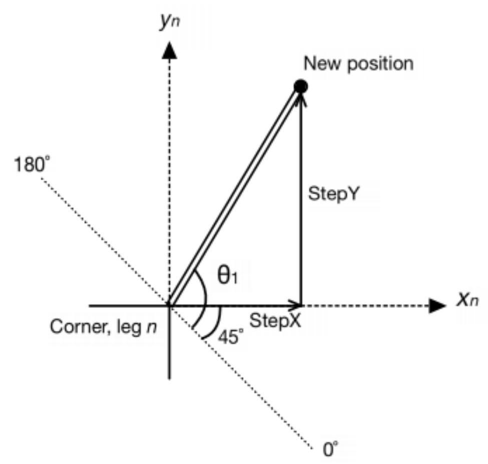


Figure 3.5. Leg seen from above, illustrating the calculation of θ_1, made by the authors.

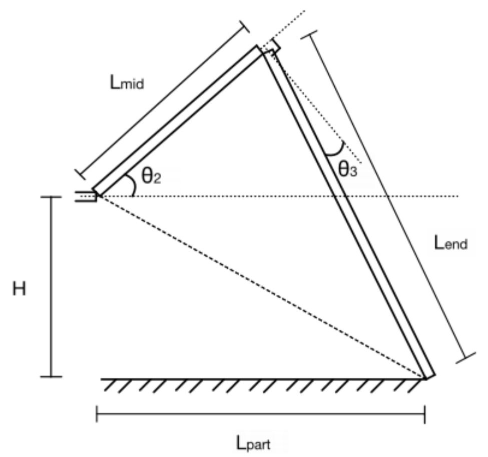


Figure 3.6. Leg seen from the side, illustrating the calculation of θ_2 and θ_3, made by the authors.

$$\theta_1 = 45^\circ + \arctan(\frac{StepY}{StepX}) \tag{3.1}$$

The angle θ_2 for the mid joint could be calculated using equation 3.2 and the angle θ_3 of the end joint was calculated using equation 3.3.

$$\theta_2 = \arccos(\frac{L_{part}^2 + H^2 + L_{mid}^2 - L_{end}^2}{2\sqrt{L_{part}^2 + H^2 a}}) - \tan(\frac{H}{L_{part}}) \tag{3.2}$$

$$\theta_3 = \frac{\pi}{2} - \arccos(\frac{L_{mid}^2 + L_{end}^2 - L_{part}^2 - H^2}{2L_{mid}L_{end}}) \tag{3.3}$$

This information is sent to another function which interpolates the servo angles in relation to the previous position and the next position. In order to do achieve this, the code uses the Arduino library *Ramp.h* [10], which contains multiple functions for interpolation. One of the main functions used in the code is **go(newvalue, rampduration, rampmode, loopmode)**, which takes a value to interpolate to and the desired duration of the interpolation. If the *ramp* variable is set to the current position of a motor, this function can be implemented to interpolate from the current position to a new one. Thus, enabling a very smooth motion by continually using the *Servo.h* [11] libraries *write* function. This setup enables easy control of the movement, as it only requires a desired height of the robots centre plate relative to the ground and the next position of the foot relative to its neutral resting position. However, moving a leg requires that the robot does not place any weight on it. Therefore, the center of mass has to be slightly shifted away from the leg that moves, so that it is in the area supported by the other three legs, as in figure 3.7. This concludes the singular movement of each foot. However, to achieve movement

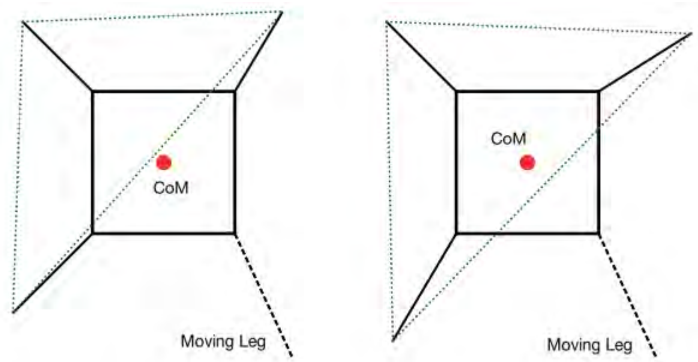


Figure 3.7. The center of mass's movement to accommodate lifting the bottom right leg, illustrated by the authors.

the robot also has to adjust its center of mass to move in the desired direction. In the first program this was implemented by twisting each leg without lifting it, thus propelling the robots center forward.

The second program expanded upon the first, introducing user input through the serial monitor and walking in four directions. The ability to move in different directions was accomplished using different cases of leg movement, depending upon the users input. The robot then took a predefined step towards the walking direction. This program was dependant on an initial positioning of the legs according to the desired walking direction before moving.

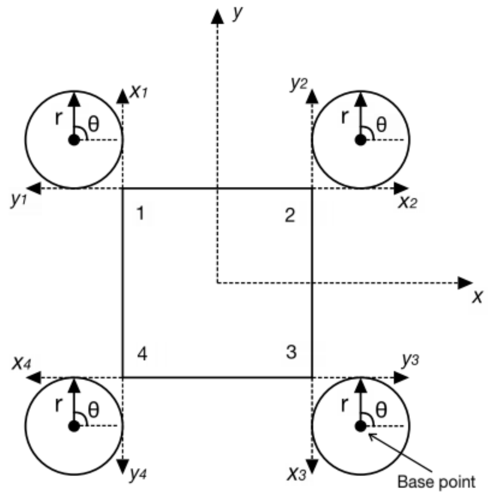


Figure 3.8. Coordinate systems for the robot, illustrated by the authors.

The third and final program implemented omnidirectional movement using a circular coordinate system, shown in figure 3.8, as well as a more stable adjustment of the center of mass. Using a circular coordinate system it is possible to define the same movement for each leg relative to a predefined center position, relative to the robot, symmetrically at each corner. The movement is defined relative to the center point of the circular system then transformed to the coordinate system of each leg using the equation 3.4.

$$\begin{bmatrix} X_n \\ Y_n \end{bmatrix} = \begin{bmatrix} BasePoint \\ BasePoint \end{bmatrix} + T_n \begin{bmatrix} X \\ Y \end{bmatrix}, \tag{3.4}$$

where T_n is the transformation matrix for each corner n as shown in table 3.1.

Table 3.1. Table of coordinate conversions.

Corner 1	**Corner 2**	**Corner 3**	**Corner 4**
$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$

Sending the location in a coordinate system specific for each leg enables the leg to move in the desired direction of movement, independent of its relative position to the body of the robot. The limits upon this movement are set by the maximum and minimum radius of the circular system. The maximum is defined by the longest step away from the robot possible. The minimum is defined by how close to its body the robot can move its legs. To handle omnidirectionality, the circular system is centered between the longest and shortest step, so that it can move an equal distance independent of the relative direction. The overlap of all the legs movement areas relative to the robots body, in figure 3.9 shows that a circle is a good match relative to its complexity.

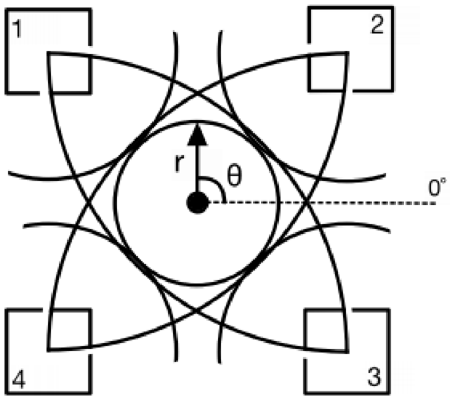


Figure 3.9. The overlap of the available area of movement for each of the four corners, and the circle of movement defined for omnidirectionality, illustrated by the authors.

To achieve smooth movement the center adjustment was changed to include a simultaneous ramp for all legs. Additionally the changes in angles for the legs were adjusted so that they would shift the robot in the desired direction. This was achieved by modifying the order the legs moved. The program recognizes the forward direction and defines each leg according to its relative position to the direction using a switch case. At first the leg pointing closest to the direction of the movement and the opposite leg move until their positions are at the front of their movement field defined the circular coordinates. Then the center of mass is moved forward by adjusting the positions of all legs. The two legs that moved before are returned to the base point, and the two other legs end up at the back of their movement field. The actual position of the robots feet on the ground does not change when moving the center this way, so it is the movement of the center of mass that has changed their position relative to their base position. The robot then repeats this process with the other two legs. The entire step cycle is demonstrated in 3.10 and the flowchart for this movement is illustrated in figure 3.11.

Because the base stance of the robot is to have every leg on the base point the robot needs to return to its base stance if the direction changes so much that it switches case. The robot will assume its base stance when it does not receive input from the serial monitor. If there is no pause between the inputs the robots movement may not entirely be in the right direction for the first step in the new

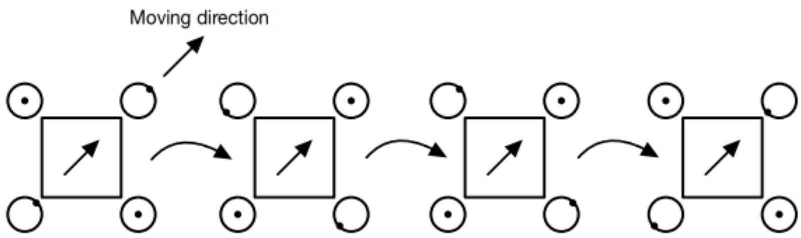


Figure 3.10. Illustration of one step sequence, made by the authors.

direction. Additionally the third program adjusts the center shift during movement, shown in figure 3.7, so that it is relative to the desired height. This means that the robot will adjust an equal amount no matter the height. This means the robot is more stable than earlier at lower heights, and retains the same stability at other heights.

3.4 Hardware and electrical circuit

The electrical hardware used in this project consisted of the Arduino UNO and twelve micro servo motors. These are connected according to the diagram in 3.12. The servo motors are powered by four AAA batteries, as that gives the highest tolerable voltage. The arduino is powered by another computer which also controls the robots movement through inputs to the Serial monitor.

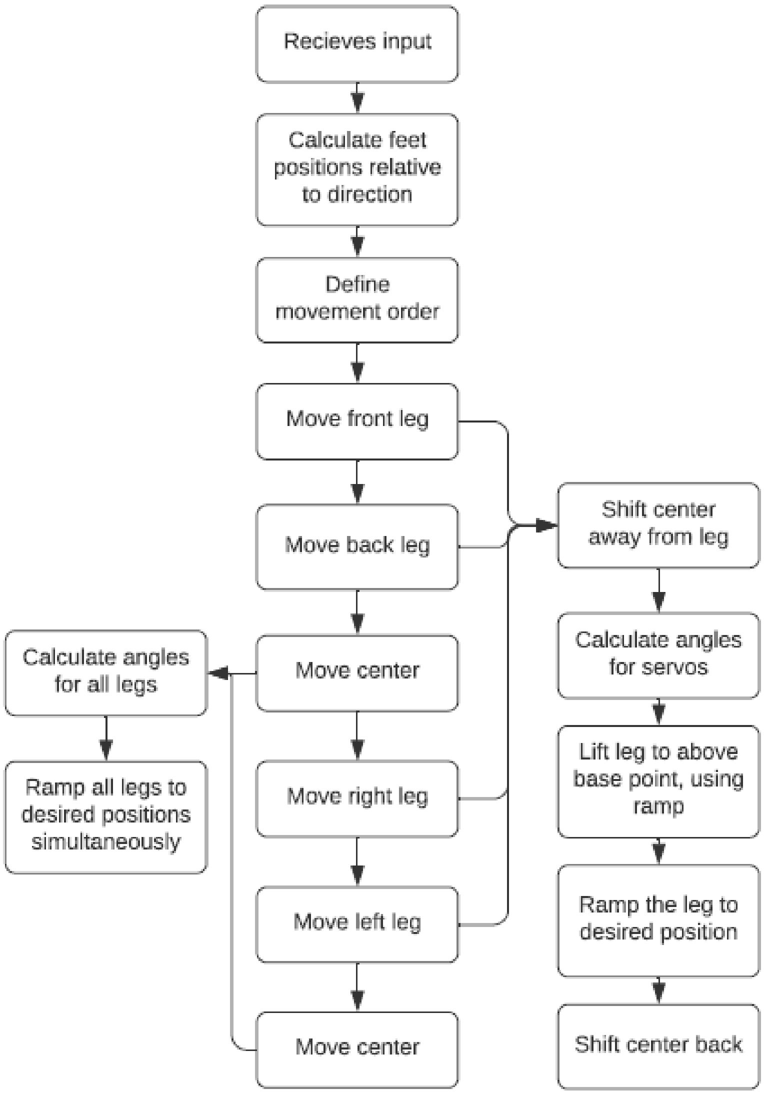


Figure 3.11. Flowchart for a full step cycle in the final program, made with Lucid[12]

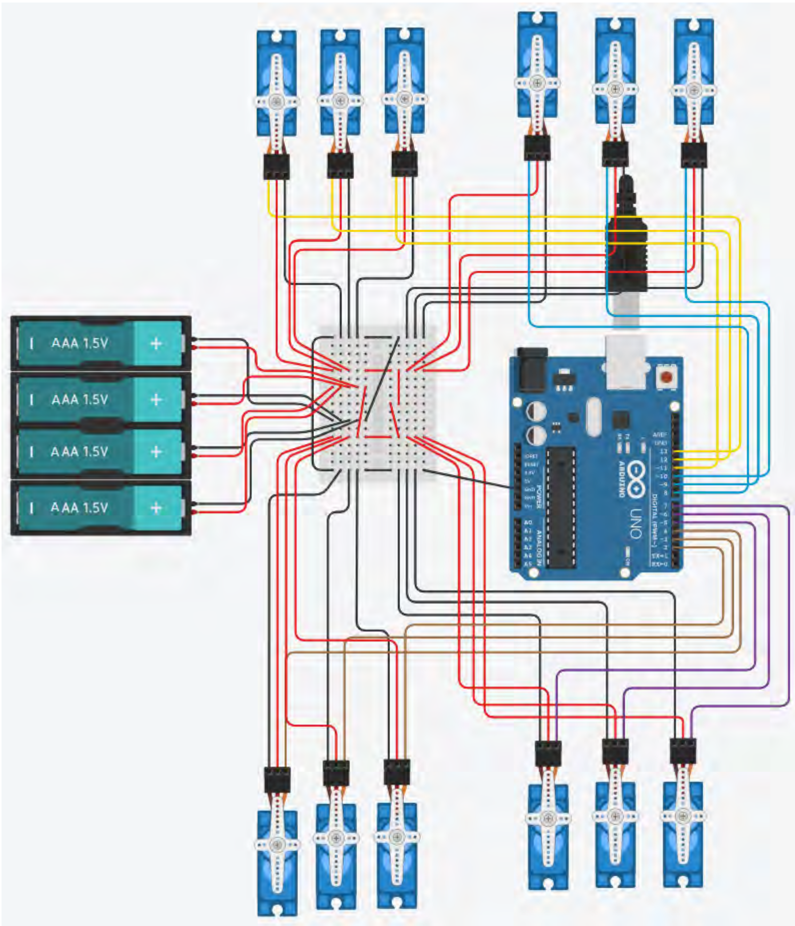


Figure 3.12. Circuit created in tinkercad [13]

Chapter 4

Results

The combination of the final prototype and code was a robot that could move smoothly in any direction, and upside down. However, the legs appear to handle movement perpendicular to their base state better than movement parallel to it. The robot was able to walk when the base point, standard height and step radius where within reasonable values. The tests showed that the best base height was at 60mm. With an increased step radius the robot could walk further. Unfortunately it was limited by the power of the servo motors, as they could not counteract the momentum upon the foot. If this limit was exceeded the robot collapses, and will in most cases not be able to right itself. Lower height means the robots legs handle collapsing better, as their believed ground height is closer to the actual ground height.

The omnidirectional movement area was most limited by diagonal movement as seen in figure 3.10. This is because the robots legs move closer or further away from the body, rather than more side to side relative to the body.

To test different speeds, the robot was measured while taking ten steps in the 270° direction, at different interpolation times.

Table 4.1. Length walked after 10 steps of 25 in direction 270°.

Ramp	100	250	400
cm	14	28	31

To test the quality of the omnidirectionallity, tests of how far the robot moved with the same step length in several directions were performed.

Table 4.2. Length walked after 15 steps of 20 in several directions, Ramp = 250 .

Direction	22.5°	45°	90°
cm	42	51	38

Chapter 5

Discussion and Conclusion

5.1 Test results

The results of the test of different interpolation time show that faster movements leads to shorter distance traveled, but between $250ms$ and $400ms$ the difference is not substantial. From visual observation of the test it is clear that this is dependent on the amount of friction between the feet and the floor tested on. Thanks to the faster relative movement on the surface, the grip is lower and therefor results in shorter steps. To counter this we could have tried different surfaces, as well as find a better way to increase the grip of each foot.

When testing different angular directions we saw that the legs appear to handle movement perpendicular to their base state better than movement parallel to it. We are not sure exactly why this is the case because the step radius was the same for all the test, but our theory is that thanks to the different positioning patterns of the legs, the shift of the center off mass is more or less effective.

5.2 Base height and its effect on the robot

If the center of mass is out of the area where it is stable, its fall will accelerate faster at higher heights, because of the increased momentum of a longer rotational arm. This means that a robot will experience a larger force upon the supporting leg when the base height is higher. This means that the robot can shift its center of mass further at lower heights, while still maintaining the same level of stability.

5.3 Movement over rough terrain.

Because the robot has no sense of what is around it, it has no ability to correct for rough terrain. However, if the robots capabilities were expanded to sense the terrain around it, it could be adjusted to be able to walk across rough terrain. One way to do this would be to use the overlap of available stepping points in the terrain and the area where the robot steps.

5.4 Implementation of wireless control.

The robot is controlled by a serial monitor, and its steps are defined by angle relative to the robot and step size, as well as whether it is upside down or not. This could easily be adapted in to a wireless connection where the robot receives the needed inputs from a Bluetooth connected device instead of the Serial Monitor. A step in one direction would for example only need the angle of direction, a step speed and knowledge of if it is upside down. The robot would then executes one full step cycle in that direction, before receiving a command again. In the code, this would simply be a change of where the input comes from. However it would require a Bluetooth module for the Arduino. Due to limitations stated in the scope this was not implemented for the robot during the project, but its a possible future development.

5.5 Gaits

To make the robot walk, there are many more possible gaits than the one that was implemented. Ranging from highly dynamic to statically stable, these could increase the robots speed, if implemented well. Due to limitations, primarily time, gait testing was very limited. We could only make the robot walk well and omni-directional using a slow stable gait. The robot could be improved by testing over values between fully dynamic gait and fully stable gait. This would give the highest speed at which the robot could still move reliably. An additional factor for the speed is the correlation between the size of steps and how fast they are taken.

5.6 Accuracy

With the available resources it is not possible to achieve incredible accuracy during movement. In part this is due to the fact that the components did not fit perfectly together. The servo arms are slightly loose, all components are assembled by hand, and servo motors do not have 100% accuracy. These discrepancies amount to a loss of accuracy that is noticeable in the robots movement. Some of the discrepancies have been counteracted in the code, for example adjusting all values for the angle of a certain motor by a small degree because it was slightly crooked. However, the design of the robot could be improved to be less loose and give a more stable movement.

A problem while testing was that the robots feet often slipped on the surface it walked on. Though this was mostly a problem at lower interpolation times, it might still have affected the tests run on higher interpolation times. Additionally the battery power might have had an effect on the results, as the robot seemed to get slower towards the end of testing. Also the human error and the suitable measurement instruments available limited the precision of the measurements. Taking more steps while testing each angle would increase the reliability of the measurements, but the current amount still gives relatively reliable results.

5.7 Conclusion

To solve the first research question, how to make the robot independent of what direction is up and down, the legs were built symmetrically. Additionally, the software was adjusted to mirror some parts of the movement to achieve the same movement independent of which side was up.

To solve the second research question, how can a four legged robot be constructed to make it omnidirectional, a circular movement area was defined for each leg, so that the steps would be the same no matter the direction. The omnidirectional movement area was most limited by diagonal movement as seen in figure 3.10. This is because the robots legs move closer or further away from the body, rather than more side to side relative to the body.

To explore the third research question, what "gait" has an optimal stability to speed ratio, the tests show that the robots movement is most reliable when taking steps within the radius of 25mm, with a leg motion time of around 250ms per move.

5.8 Future work

Improvements that could be made upon the robot include the implementation of wireless control, as discussed above. Additionally the robot could be optimized in many ways. The 3D model could be optimized for weight. The parts of the legs could be optimized so that the horizontally moving part is closer to the center, thus lessening the load upon the joints. Additionally, the length of the middle and end parts could be adjusted so that they could handle the momentum of full extension.

The robot could also be optimized in its movement. This could be done with extensive testing, as only coarse tests were made for several parts of the system. The omnidirectional circular area could be exchanged for an area better fitting to the available overlap displayed in 3.9. However, this would make the robot slightly dependent of what direction it is walking, making it a trade-off between omnidirectionality and speed.

Bibliography

[1] Satoshi Kitano et al. “Development of lightweight sprawling-type quadruped robot TITAN-XIII and its dynamic walking”. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 6025–6030. DOI: 10.1109/IROS.2013.6697231.

[2] Arduino.cc. *ARDUINO UNO REV3*. URL: https://store.arduino.cc/arduino-uno-rev3. (accessed: 03.02.2021).

[3] Miguel Gudino. *Introduction to Microcontrollers*. URL: https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller. (accessed: 03.02.2021).

[4] Kjell & Company. *Luxorparts SG90 Micro-servo 4-pack*. URL: https://www.kjell.com/se/produkter/el-verktyg/arduino/arduino-tillbehor/luxorparts-sg90-micro-servo-4-pack-p90720. (accessed: 04.02.2021).

[5] Jameco Electronics. *How Servo Motors Work*. URL: https://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html. (accessed: 04.02.2021).

[6] Xilun Ding Kun Xu Peijin Zi. “Gait Analysis of Quadruped Robot Using the Equivalent Mechanism Concept Based on Metamorphosis”. In: *Chin. J. Mech. Eng* 32.8 (2019). URL: https://doi.org/10.1186/s10033-019-0321-2.

[7] Rickard Nilsson. “Inverse Kinematics”. Master Thesis, Luleå Univerity of Technology, (2009). URL: https: //www.diva-portal.org/smash/get/diva2:1018821/FULLTEXT01.pdf.

[8] Seimens. *Solid Edge*. URL: https://solidedge.siemens.com/en/. (accessed: 21.02.2021).

[9] Electrokit. *Servo MS-1.3-9*. URL: https://www.electrokit.com/produkt/servo-ms-1-3-9/. (accessed: 04.02.2021).

[10] Sylvain Garnavault. *ramp.h*. Sept. 4, 2020. URL: https://github.com/siteswapjuggler/RAMP. (accessed: 01.05.2021).

[11] Martino Facchim. *servo.h*. Sept. 4, 2020. URL: https://github.com/arduino-libraries/Servo. (accessed: 21.05.2021).

[12] Lucid. *Diagram your people, processes, and systems*. URL: https://www.lucidchart.com/pages/product. (accessed: 06.5.2021).

[13] Autodesk. *Tinkercad*. URL: `https://www.tinkercad.com/`. (accessed: 20.02.2021).

Appendix A

Arduino UNO datasheet

Atmel

ATmega328P

8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash

DATASHEET

Features

- High performance, low power AVR® 8-bit microcontroller
- Advanced RISC architecture
  - 131 powerful instructions – most single clock cycle execution
  - 32 × 8 general purpose working registers
  - Fully static operation
  - Up to 16MIPS throughput at 16MHz
  - On-chip 2-cycle multiplier
- High endurance non-volatile memory segments
  - 32K bytes of in-system self-programmable flash program memory
  - 1Kbytes EEPROM
  - 2Kbytes internal SRAM
  - Write/erase cycles: 10,000 flash/100,000 EEPROM
  - Optional boot code section with independent lock bits
    - In-system programming by on-chip boot program
    - True read-while-write operation
  - Programming lock for software security
- Peripheral features
  - Two 8-bit Timer/Counters with separate prescaler and compare mode
  - One 16-bit Timer/Counter with separate prescaler, compare mode, and capture mode
  - Real time counter with separate oscillator
  - Six PWM channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature measurement
  - Programmable serial USART
  - Master/slave SPI serial interface
  - Byte-oriented 2-wire serial interface (Phillips I^2C compatible)
  - Programmable watchdog timer with separate on-chip oscillator
  - On-chip analog comparator
  - Interrupt and wake-up on pin change
- Special microcontroller features
  - Power-on reset and programmable brown-out detection
  - Internal calibrated oscillator
  - External and internal interrupt sources
  - Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby, and extended standby

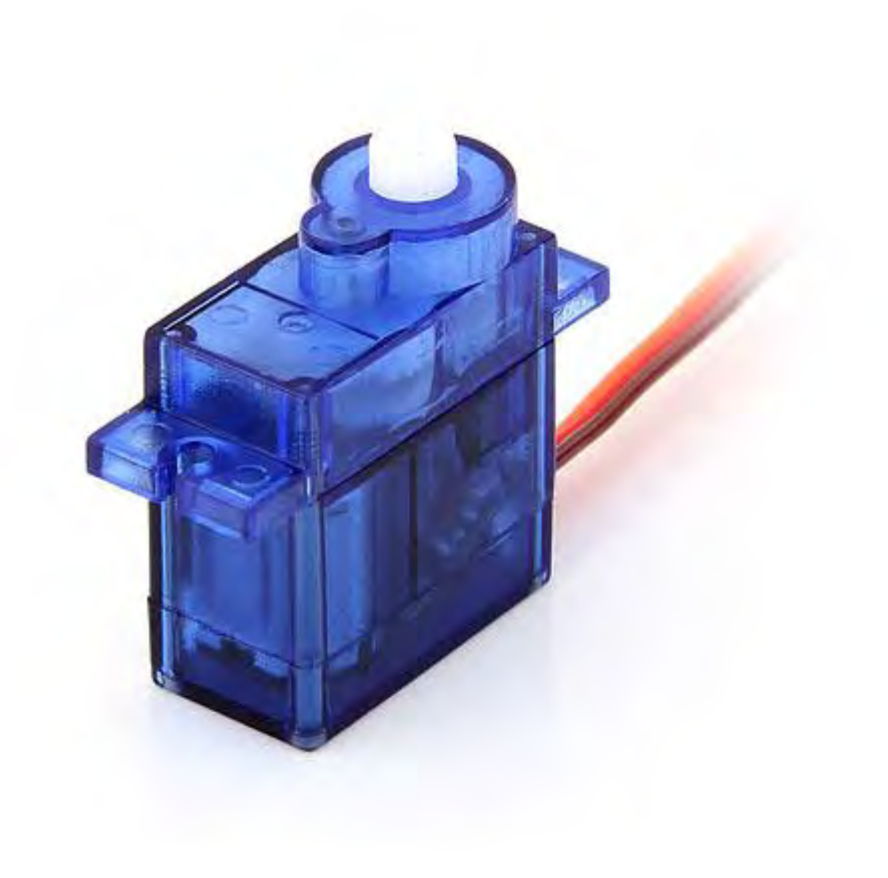
- I/O and packages
  - 23 programmable I/O lines
  - 32-lead TQFP, and 32-pad QFN/MLF
- Operating voltage:
  - 2.7V to 5.5V for ATmega328P
- Temperature range:
  - Automotive temperature range: –40°C to +125°C
- Speed grade:
  - 0 to 8MHz at 2.7 to 5.5V (automotive temperature range: –40°C to +125°C)
  - 0 to 16MHz at 4.5 to 5.5V (automotive temperature range: –40°C to +125°C)
- Low power consumption
  - Active mode: 1.5mA at 3V - 4MHz
  - Power-down mode: 1µA at 3V

Appendix B

MS-1.3-9 Servo motor datasheet

MS-1.3-9

Servo Motor MS-1.3-9



- Dimensions: 23.2 x 12.5 x 22 mm
- Operating Speed: 0.12sec/60degree (4.8V), 0.10sec/60degree (6V)
- Stall Torque: 1.3kg.cm/18.09oz.in(4.8V)
- Operating Voltage: 4.8V~6V
- Control System: Analog
- Direction: CCW
- Operating Angle: 120degree
- Required Pulse: 900us-2100us
- Bearing Type: None
- Gear Type: Plastic
- Motor Type: Metal
- Connector Wire Length: 20 cm

Appendix C

Arduino code

```
//Omnidirectional quadruped sprawling type robot
//Bachelors Thesis MF133X
//Date: 2021-05-21
//Authors: Samuel Stenow, Simon Lindenfors
//Description: Code running on Arduino Uno.
//Recieves input from the serial monitor
//Translates inputs into movement

#include <Servo.h>
#include <Ramp.h>


Servo End_One; //End servo of leg one
Servo Mid_One; //Mid servo of leg one
Servo First_One; //First servo of leg one

Servo End_Two; //End servo of leg Two
Servo Mid_Two; //Mid servo of leg Two
Servo First_Two; //First servo of leg Two

Servo End_Three; //End servo of leg Three
Servo Mid_Three; //Mid servo of leg Three
Servo First_Three; //First servo of leg Three

Servo End_Four; //End servo of leg Four
Servo Mid_Four; //Mid servo of leg Four
Servo First_Four; //First servo of leg Four

double L_mid = 55; //Length of mid part
double L_end = 80; //Length of end part
double L_first = 9 + 18; //Length of first part
double StepRadius = 20; //Radius of circle which foot can move within
double Base_point = 50; //centrpoint of the cirkle (x and y distance
    from the first servo to this point)
double Base_height = 60; //the centre parts base height from ground

int i;
```

```
int j;

double StandardAngle = 90; //Standard angle for mid and end servo
double v_First; //Angle of first servo
double v_Mid; //Angle of mid servo
double v_End; // Angle of end servo

double L_hyp;
double L_part; //partial lenght from first servo to foot position without length of first part

String input; //input from serial monitor
int intInput; // input from serial monitor converted to integer

struct Leg{ //Struct for every leg

  String Name; //Name of the leg
  Servo First; //servo of first joint
  Servo Mid; //servo of mid joint
  Servo End; //servo of end joint

  double midCorrection; //small angle correction of mid joint
  double endCorrection; //small angle correction of end joint

  double First_Pos; //current angle position of first joint
  double Mid_Pos; //current angle position of mid joint
  double End_Pos; //current angle position of end joint

  double totX; //
  double totY; //

  ramp First_Ramp; //ramp variable for first joint
  ramp Mid_Ramp; //ramp variable for mid joint
  ramp End_Ramp; //ramp variable for end joint

  double v_First_Deg; //new angle for first joint
  double v_Mid_Deg; //new angle for first joint
  double v_End_Deg; //new angle for first joint
};

struct Leg LegOne,LegTwo,LegThree,LegFour; //creating struct for all four legs


void setup()
{
  Serial.begin(9600);
  End_One.attach(11); //Tilldelar varje servo en port
  Mid_One.attach(12);
  First_One.attach(13);

  End_Two.attach(8);
  Mid_Two.attach(9);
  First_Two.attach(10);
```

```

End_Three.attach(2);
Mid_Three.attach(3);
First_Three.attach(4);

End_Four.attach(5);
Mid_Four.attach(6);
First_Four.attach(7);


//Asigning every struct variable to each leg
LegOne.Name = "Leg One";
LegOne.First = First_One;
LegOne.Mid = Mid_One;
LegOne.End = End_One;
LegOne.First_Pos = 90;
LegOne.Mid_Pos = 90;
LegOne.End_Pos = 90;
LegOne.midCorrection =  - 10;
LegOne.endCorrection =  - 10;
LegOne.First_Ramp = 0;
LegOne.Mid_Ramp = 0;
LegOne.End_Ramp = 0;
LegOne.v_First_Deg = 0;
LegOne.v_Mid_Deg = 0;
LegOne.v_End_Deg = 0;

LegTwo.Name = "Leg Two";
LegTwo.First = First_Two;
LegTwo.Mid = Mid_Two;
LegTwo.End = End_Two;
LegTwo.First_Pos = 90;
LegTwo.Mid_Pos = 90;
LegTwo.End_Pos = 90;
LegTwo.endCorrection = 5;
LegTwo.First_Ramp = 0;
LegTwo.Mid_Ramp = 0;
LegTwo.End_Ramp = 0;
LegTwo.v_First_Deg = 0;
LegTwo.v_Mid_Deg = 0;
LegTwo.v_End_Deg = 0;

LegThree.Name = "Leg Three";
LegThree.First = First_Three;
LegThree.Mid = Mid_Three;
LegThree.End = End_Three;
LegThree.First_Pos = 90;
LegThree.Mid_Pos = 90;
LegThree.End_Pos = 90;
LegThree.First_Ramp = 0;
LegThree.Mid_Ramp = 0;
LegThree.End_Ramp = 0;
LegThree.v_First_Deg = 0;
LegThree.v_Mid_Deg = 0;
```

```
  LegThree.v_End_Deg = 0;

  LegFour.Name = "Leg Four";
  LegFour.First = First_Four;
  LegFour.Mid = Mid_Four;
  LegFour.End = End_Four;
  LegFour.First_Pos = 90;
  LegFour.Mid_Pos = 90;
  LegFour.End_Pos = 90;
  LegFour.endCorrection = -20;
  LegFour.First_Ramp = 0;
  LegFour.Mid_Ramp = 0;
  LegFour.End_Ramp = 0;
  LegFour.v_First_Deg = 0;
  LegFour.v_Mid_Deg = 0;
  LegFour.v_End_Deg = 0;
}

void TurnOff(){
  End_One.detach(); //disconnecting every servo from its signal port
  Mid_One.detach();
  First_One.detach();

  End_Two.detach();
  Mid_Two.detach();
  First_Two.detach();

  End_Three.detach();
  Mid_Three.detach();
  First_Three.detach();

  End_Four.detach();
  Mid_Four.detach();
  First_Four.detach();
}

//-------------------------------------------------

void Interpolate (struct Leg * leg, int Time){ //interpolates the
    given values using go function in Ramp.h
  leg -> First_Ramp = leg -> First_Pos;
  leg -> Mid_Ramp = leg -> Mid_Pos;
  leg -> End_Ramp = leg -> End_Pos;
  leg -> First_Ramp.go(leg -> v_First_Deg, Time);
  leg -> Mid_Ramp.go(leg -> v_Mid_Deg + leg -> midCorrection, Time);
  leg -> End_Ramp.go(leg -> v_End_Deg + leg -> endCorrection, Time);
}

//-------------------------------------------------

void Write (struct Leg * leg){ //writes the new position for each
    servo
  leg -> First.write(leg -> First_Ramp.update());
  leg -> Mid.write(leg -> Mid_Ramp.update());
```

```
  leg -> End.write(leg -> End_Ramp.update());
}

//-----------------------------------------------------

void update_position (struct Leg * leg){ //updates the positions of
    each servo
  leg -> First_Pos = leg -> First_Ramp.update();
  leg -> Mid_Pos = leg -> Mid_Ramp.update();
  leg -> End_Pos = leg -> End_Ramp.update();
}

//-----------------------------------------------------

void Move (struct Leg * leg){ //calls for interpolation, write of
    servos and updates their position

  Interpolate(leg, 250);

  while(leg -> First_Ramp.isRunning()){
    Write(leg);
  }

  update_position(leg);
}

//-----------------------------------------------------

//calculate every new angle for joints
void Calculate_angles (struct Leg* leg, double Height, double StepX,
    double StepY, bool upside_down, bool simultaneously){

  v_First = PI/4 + atan(StepY/StepX); //calculates angle for first
    servo

  L_part = sqrt(pow(StepY,2) + pow(StepX,2)) - L_first;
  L_hyp = sqrt(pow(Height,2) + pow(L_part,2));

  v_Mid = acos((pow(L_hyp,2) + pow(L_mid,2) - pow(L_end,2))/(2*L_hyp*
    L_mid)) - atan(Height/L_part); //calculates angle of mid servo
  v_End = PI/2 - acos((pow(L_mid,2) + pow(L_end,2) - pow(L_hyp,2))
    /(2*L_mid*L_end)); //calculates angle of end servo


  if (upside_down &! simultaneously){ //if the robot is upside down
    and only one leg is suppose to move
    leg -> v_First_Deg = v_First * (180/PI); //saves angle to struct
    leg -> v_Mid_Deg = StandardAngle - v_Mid * (180/PI);
    leg -> v_End_Deg = StandardAngle - v_End * (180/PI);
    Move(leg);
  }
  else if (simultaneously &! upside_down){ //if robot is suppose to
    move all legs simultaniously and is not upside down
    leg -> v_First_Deg = v_First * (180/PI);
```

```
    leg -> v_Mid_Deg = StandardAngle + v_Mid * (180/PI);
    leg -> v_End_Deg = StandardAngle + v_End * (180/PI);
  }
  else if (simultaneously && upside_down){ //if robot is suppose to
    move all legs simultaniously and is upside down
    leg -> v_First_Deg = v_First * (180/PI);
    leg -> v_Mid_Deg = StandardAngle - v_Mid * (180/PI);
    leg -> v_End_Deg = StandardAngle - v_End * (180/PI);
  }
  else{ // if robot is suppose to only suppose to move one leg and is
     not upside down
    leg -> v_First_Deg = v_First * (180/PI);
    leg -> v_Mid_Deg = StandardAngle + v_Mid * (180/PI);
    leg -> v_End_Deg = StandardAngle + v_End * (180/PI);
    Move(leg);
  }
}

//---------------------------------------------------

void Move_all_legs_at_once ( struct Leg * Front, struct Leg * Back,
    struct Leg * Left, struct Leg * Right){

  Interpolate(Front, 250);
  Interpolate(Back, 250);
  Interpolate(Left, 250);
  Interpolate(Right, 250);

  while(Front -> First_Ramp.isRunning()){
    Write(Front);
    Write(Back);
    Write(Left);
    Write(Right);
  }

  update_position(Front);
  update_position(Back);
  update_position(Left);
  update_position(Right);

}

//---------------------------------------------------

//calls for functiones that together moves the center of mass forward
void Full_center_move(struct Leg * First_to_Base_point, struct Leg *
    Second_to_Base_point, struct Leg * First_to_move_back, struct Leg
    * Second_to_move_back, bool upside_down){
  Calculate_angles(First_to_Base_point, Base_height, Base_point,
    Base_point, upside_down, true);
  Calculate_angles(Second_to_Base_point, Base_height, Base_point,
    Base_point, upside_down, true);

  Calculate_angles(First_to_move_back, Base_height, 2*Base_point -
```

```
    First_to_move_back -> totX, 2*Base_point - First_to_move_back -> totY, upside_down, true);
  Calculate_angles(Second_to_move_back, Base_height, 2*Base_point - Second_to_move_back -> totX, 2*Base_point - Second_to_move_back -> totY, upside_down, true);

  Move_all_legs_at_once(First_to_Base_point,Second_to_Base_point,First_to_move_back,Second_to_move_back);
}

//--------------------------------------------------

void Center_shift (struct Leg * Opposite,struct Leg * Left, struct Leg * Right,struct Leg * Corner, int ToOrFrom){ //
  for(int i = 0; i <= 2; i ++ ){
    Opposite -> Mid.write(Opposite-> Mid_Pos - 2*ToOrFrom);
    Opposite -> Mid_Pos -= (7-6*(Base_height/100))*ToOrFrom; // Depending on base height, the amount of correction changes,
    Opposite -> End.write(Opposite-> End_Pos - 2*ToOrFrom);  //lower base height results in greater angular correction
    Opposite -> End_Pos -= (7-6*(Base_height/100))*ToOrFrom;
    Left -> First.write(Left-> First_Pos - 2*ToOrFrom);
    Left -> First_Pos -= (12-8*(Base_height/100))*ToOrFrom;
    Right -> First.write(Right-> First_Pos - 2*ToOrFrom);
    Right -> First_Pos += (12-8*(Base_height/100))*ToOrFrom;
    delay(100);
  }

}

//--------------------------------------------------

// decides depending on which leg is moving, which legs that are suppose to shift the center of mass
void Center_shift_order(struct Leg * leg, int ToOrFrom){
  if (leg -> Name == "Leg One"){
    Center_shift(&LegThree, &LegTwo, &LegFour, &LegOne, ToOrFrom); // calls for the center off mass shifting
    Serial.println("LegOne");
  }
  else if(leg -> Name == "Leg Two"){
    Center_shift(&LegFour, &LegThree, &LegOne, &LegTwo, ToOrFrom);
    Serial.println("LegTwo");
  }
  else if(leg -> Name == "Leg Three"){
    Center_shift(&LegOne, &LegFour, &LegTwo, &LegThree, ToOrFrom);
    Serial.println("Leg Three");
  }
  else{
    Center_shift(&LegTwo, &LegOne, &LegThree, &LegFour, ToOrFrom);
    Serial.println("Leg Four");
  }
}

```

```
//-----------------------------------------------------

void Full_step_one_leg(struct Leg * leg, bool upside_down, bool simultaneously){ //cycle of one leg move
  Center_shift_order(leg, - 1);
  Calculate_angles(leg, Base_height - 40, Base_point, Base_point, upside_down, simultaneously);
  Calculate_angles(leg, Base_height, leg -> totX, leg -> totY, upside_down, simultaneously);
  Center_shift_order(leg,1);
}

//-----------------------------------------------------

//moves legs one at a time in the correct order in correlation to direction
void Move_order(struct Leg * Front, struct Leg * Back, struct Leg * Right, struct Leg * Left, bool upside_down){

  Full_step_one_leg(Front, upside_down, false); //move front leg relative to direction
  delay(1000);
  Full_step_one_leg(Back, upside_down, false); //move back leg relative to direction
  delay(1000);
  Full_center_move(Front, Back, Right, Left, upside_down); //shift centre of mass in forward direction
  delay(1000);

  Full_step_one_leg(Right, upside_down, false); //move right leg relative to direction
  delay(1000);
  Full_step_one_leg(Left, upside_down, false); //move left leg relative to direction
  delay(1000);
  Full_center_move(Front, Back, Right, Left, upside_down); //shift centre of mass in forward direction
  delay(1000);
}

//-----------------------------------------------------

//calculates positions for the foot that is suppose to move and desides order depending on direction
void Full_step_cycle(double Direction, double Speed, bool upside_down){
  double x = StepRadius*cos(Direction)*Speed;
  double y = StepRadius*sin(Direction)*Speed;


  LegOne.totX = Base_point + y;
  LegTwo.totX = Base_point + x;
  LegThree.totX = Base_point - y;
  LegFour.totX = Base_point - x;
```

```

  LegOne.totY = Base_point - x;
  LegTwo.totY = Base_point + y;
  LegThree.totY = Base_point + x;
  LegFour.totY = Base_point - y;

  if(0<=Direction && Direction<=PI/2){
    Move_order(&LegTwo, &LegFour, &LegThree, &LegOne, upside_down);
  }
  if(PI/2<Direction && Direction<=PI){
    Move_order(&LegOne, &LegThree, &LegTwo, &LegFour, upside_down);
  }
  if(PI<Direction && Direction<=3*PI/2){
    Move_order(&LegFour, &LegTwo, &LegOne, &LegThree, upside_down);
  }
  if(3*PI/2<Direction && Direction<=2*PI){
    Move_order(&LegThree, &LegOne, &LegFour, &LegTwo, upside_down);
  }
}

//--------------------------------------------------

void loop()
{
  if(Serial.available()){
    input = Serial.readStringUntil('\n');
    intInput = input.toInt();
    Serial.print("You typed: ");
    Serial.println(intInput);
  }
  //folling are tests of different directions and steps
  switch (intInput){
    case 7:
      TurnOff();
      delay(1000);
      intInput == 0;
      break;
    case 3:
      Center_shift_order(&LegOne,-1);

      Calculate_angles(&LegOne,50,30,30,false,false);
      delay(500);
      Calculate_angles(&LegOne,20,80,80,false,false);
      delay(500);
      Calculate_angles(&LegOne,50,30,30,false,false);
      delay(500);
      Calculate_angles(&LegOne,20,80,80,false,false);
      intInput=0;
      break;
    case 2:
      Full_step_cycle(0,1,false);
      intInput = 0;
      break;
    case 4:
```

```
      Full_step_cycle(PI/2,1,false);
      intInput = 0;
      break;
    case 8:
      Full_step_cycle(PI,1,false);
      intInput = 0;
      break;
    case 6:
      Full_step_cycle(3*PI/2,1,false);
      intInput = 0;
      break;
    case 45:
      Full_step_cycle(PI/2,1,false);
      Full_step_cycle(PI/2,1,false);
      Full_step_cycle(PI/2,1,false);
      Full_step_cycle(PI/2,1,false);
      Full_step_cycle(PI/2,1,false);
      intInput = 0;
      break;
    case 9:
      Full_step_cycle(5*PI/8,1,false);
      Full_step_cycle(5*PI/8,1,false);
      Full_step_cycle(5*PI/8,1,false);
      Full_step_cycle(5*PI/8,1,false);
      Full_step_cycle(5*PI/8,1,false);
      intInput = 0;
      break;
    case 4555:
      Full_step_cycle(3*PI/2,1,true);
      Full_step_cycle(3*PI/2,1,true);
      Full_step_cycle(3*PI/2,1,true);
      Full_step_cycle(3*PI/2,1,true);
      Full_step_cycle(3*PI/2,1,true);
      Full_step_cycle(3*PI/2,1,true);
      intInput = 0;
      break;
    case 1793:
      Full_step_cycle(0,0,false);
      for(int v = 0; v<=20; v++){
        Full_step_cycle(PI*v/10,1,false);
        Full_step_cycle(PI*v/10,1,false);
      }
      intInput = 0;
      break;
    case 1452:
      for(int v = 1; v<=2; v++){
        Full_step_cycle(7*PI/4,1,false);
      }
      for(int v = 1; v<=2; v++){
        Full_step_cycle(PI,1,false);
      }
      for(int v = 1; v<=2; v++){
        Full_step_cycle(PI/2,1,false);
      }
```

```
      for(int v = 1; v<=2; v++){
        Full_step_cycle(0,1,false);
      }
      intInput = 0;
      break;
    case 666:
      Full_step_cycle(PI/4,0,false);
      intInput = 0;
      break;
  }
  delay(4000);
}
```

Appendix D

Acumen code

```
//Omnidirectional quadruped sprawling type robot
//Bachelors Thesis MF133X
//Date: 2021-05-21
//Authors: Samuel Stenow, Simon Lindenfors
//Description: Code for Acumen simulation.
//Simulates a very simple design of the robot
//and a circular movement of one leg

model Main(simulator) =
initially
c1 = create Robot((0,0,0),(0,0,0),0), //creates robot
//creates and initiates variables
theta = 0, theta' = 0, theta'' = 0,//angle, angular velocity as well as acceleration
theta2 = 0, theta2' = 0, theta2'' = 0,
l = 2, //radius
l2 = 4,
x = 0, y = 0, //x and y pos difference
x2 = 0, y2 = 0



always
if theta < pi/4 //to move a quarter of a cirle
then theta'' = 0.25, theta2'' = 0.25 //increase angular acceleration
else if theta' > 0 //stops when reaching the destination
then theta' = 0, theta2' = 0
else if theta > 0 //changing acceleration to 0
then theta'' = 0, theta2'' = 0
noelse,
//equations for x and y coordinates
x = l*cos(theta),y = l*sin(theta),
x2 = l2*cos(theta2)-2,y2 = l2*sin(theta2),
//send to robot
c1.pos1 = (x,y,0),
c1.pos2 = (x2,y2,0),
c1.theta = theta2
```

```
model Robot(pos1,pos2,theta) =
initially
_3D = (),_Plot=()
always
//skapar roboten
_3D = (Box center =(0,0,0) size = (4,7,0.5) color = cyan rotation =
    (0,0,0)
Sphere center=(2,3.5,0) size = 0.5 color=magenta rotation=(0,0,0)
Sphere center=(-2,-3.5,0) size = 0.5 color=red rotation=(0,0,0)
Sphere center=(2,-3.5,0) size = 0.5 color=green rotation=(0,0,0)
Sphere center=(-2,3.5,0) size = 0.5 color=yellow rotation=(0,0,0)
Cylinder center=pos1+(2,3.5,0) size = (3,0.2) color = black rotation
    =(0,0,theta+pi/2)
Cylinder center=(4,-3.5,0) size = (3,0.2) color = black rotation
    =(0,0,pi/2)
Cylinder center=(-4,-3.5,0) size = (3,0.2) color = black rotation
    =(0,0,pi/2)
Cylinder center=(-4,3.5,0) size = (3,0.2) color = black rotation
    =(0,0,pi/2)
Sphere center=pos2+(4,3.5,0) size = 0.5 color=magenta rotation
    =(0,0,0)
Sphere center=(-6,-3.5,0) size = 0.5 color=red rotation=(0,0,0)
Sphere center=(6,-3.5,0) size = 0.5 color=green rotation=(0,0,0)
Sphere center=(-6,3.5,0) size = 0.5 color=yellow rotation=(0,0,0)
Cylinder center=pos2+(4,3.5,-1.5) size = (3,0.2) color = black
    rotation=(pi/2,0,0)
Cylinder center=(6,-3.5,-1.5) size = (3,0.2) color = black rotation=(
    pi/2,0,0)
Cylinder center=(-6,-3.5,-1.5) size = (3,0.2) color = black rotation
    =(pi/2,0,0)
Cylinder center=(-6,3.5,-1.5) size = (3,0.2) color = black rotation=(
    pi/2,0,0)
)
```

TRITA TRITA-ITM-EX 2021:35