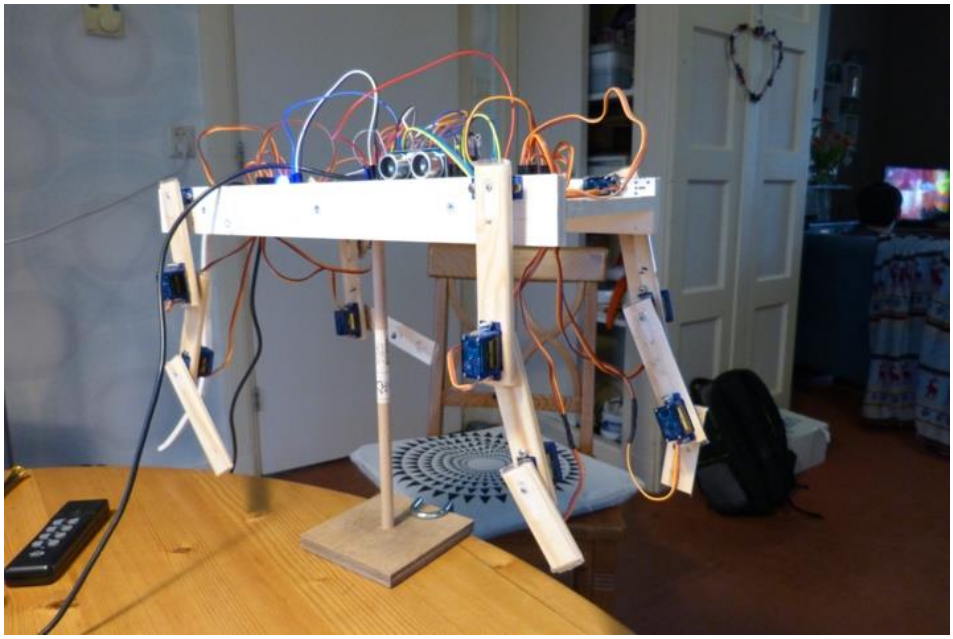
Jan
2021



Modeling animal-like movements with a robot

Profiel Werkstuk vwo Natuurkunde

MSc. L. Dappers

January 12, 2021

Arne van Iterson

Job Vonk

Milan den Besten

Table of contents

Introduction

Robots are becoming a more important aspect of our daily lives. Robots help with automating dull or labour-intensive tasks. Most of the products in our capitalistic society are – either fully or partly – manufactured by robots. Although robots are mostly used for producing goods, they can also be used for other purposes. We are very interested in programming and building electronic circuits. So, for our *Profielwerkstuk* we wanted to make a robot. We want to research how to create a robot that can walk similar to a cat. The general idea is that we film a cat walking by from the side. Then we parse the video to workable data. Then we build a cat robot by using an Arduino to manage the motors and control them to recreate a cat's walking pattern. We can create legs using wood, and we can attach the motors to a leg piece to rotate that leg piece itself. Lastly, we upload a program to the Arduino containing the data we parsed earlier that will control the motors to recreate the walking animation.

In the coming chapters, we will discuss how we

- Acquired data for the cat's walking pattern;
- Created a program that can parse and visualize that data said data;
- Created all the components needed to build the robot's structure;
- Wired the hardware;
- Programmed the hardware;
- Assembled everything into one working robot.

Research

Bio-inspired robotics

Biomimetics

Replication of the movement of animals has always been desired. Humans can learn from animals how they move through air, water and other terrain and apply these techniques to machines and robots. The application of observations of animal movements is called biomimetics and is considered a relatively new area of study in biophysics (Travers & Choset, n.d.). One of the most famous examples of biomimicry is an improved design of the bullet train, which was based on the beak of a kingfisher bird (both pictured in figure 1). Other examples include the fact that improved solar panels are based on butterfly wings that can absorb light at virtually any angle, and Velcro, which is based on the sticky hooks of the burr flowers (Alexander, 2018).

Figure 1 - The bullet train's (left) design was based on the beak of a kingfisher bird (right)

Bio-inspired robotics

Technically, our imitation of the cats' walking pattern falls under the category of biomimetics, but it shares most aspects of bio-inspired robotics. This encompasses biomimicry applied to robotics, where the copied elements are usually improved upon (Vincent, Bogatyreva, Bogatyrev, Bowyer, & Pahl, 2006). A famous company that excels in bio-inspired robotics is Boston Dynamics, which has produced many robots for businesses (Protalinski, Boston Dynamics starts selling its Spot robot — for $74,500, 2020). Examples of cases in which these robots can prove useful include reducing COVID-19 infections by reducing human contact, monitoring areas that are unsuitable for humans or unreachable by them. The Spot (pictured in figure 2), one of their most recent inventions, is known to be canine-inspired and the Atlas is inspired by humans and aims to replicate human movements as accurately as possible. The United States military also experiments with using robotic dogs made by Boston Dynamics, which they plan to use as packhorses. Two of these robots have been discontinued, however, as they made too much noise (Degeler, 2015).

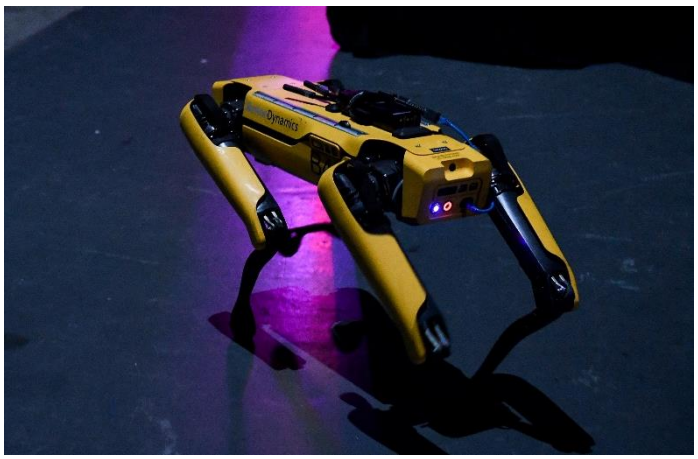


Figure 2 – Boston Dynamics' The Spot

Robotic pets

Robotic pets are rising and are gradually releasing on their previously unexplored market. These robots are often associated with the entertainment of the elderly, as they do not need to care for these dogs. They can also be deployed for children as a learning tool or for entertainment as well. People have criticised the way real pets are cared for and if it is ethical to have a pet. While the rise of robotic pets has not yet proven to correspond to the decline of actual pets, it will surely play a role.

Locomotion of cats

Cats have the same bones as humans in their limbs. Their legs don't all share the same bones, however: their forelegs consist of a humerus, radius and ulna, which appear in the arms of humans. Their hind legs do share the same bones as human legs. Their clavicle, to which the arms in humans are attached, differentiates from humans, in that it doesn't connect to any other bone which allows them to pass through very small spaces (ZooLab, 2002).

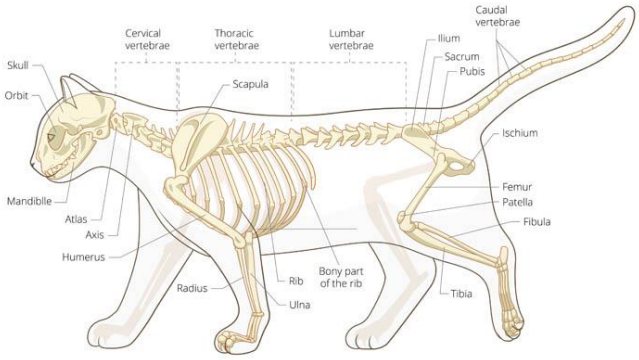


Figure 3 - The skeleton of a cat

Cats are digitigrades, which means that they walk on their toes (digits), unlike humans. Cats can also retract the claws on their paws when they are not using them. Furthermore, cats place their hind paws at the same location as their forepaws, which is a highly precise method of walking (figure 4). It also makes for great noise reduction and footprint reduction. The brain of cats is wired to do this in a process called 'direct registering'. This is also visible in *catmesh_pws* (discussed later).

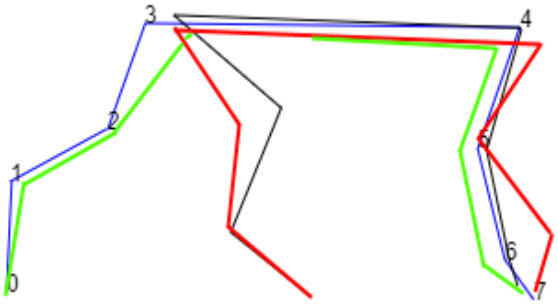


Figure 4 - Two frames of catmesh_pws overlayed to depict the direct registering phenomenon.

Building a robot

Acquiring data

To allow a robot to move based on real cats movements, a model will have to be made. There are several different ways to create a model, most of which are based on a video of the movement concerned. This video can be extended with depth data or markers on the actor of the movement to increase measurement accuracy during processing. This data will then be processed using a modelling program such as CMA Coach 7 or DMAS Suite.

Putting the data into movement

The raw data from the model will have to be processed into movement data by a microcontroller, by far the easiest and most accessible way to do this is using an Arduino compatible microcontroller. An Arduino, as it will be referred to from now on, is loaded with a bootloader which allows executing code compiled from a modified version of C++. The Arduino software allows for programming over USB and it supports various libraries for interfacing with other controllers (discussed later).

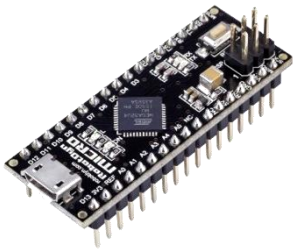


Figure 5 - Arduino Nano

As the data contained in the model is quite big, the ESP32 by Espressif Systems (figure 6) becomes a more viable option. The microcontroller powering this Arduino is an ESP32, which has a lot more memory and processing power than most Arduino compatible microcontrollers. This does come at a compromise, the ESP32 dev board we'll be using is a lot bigger than, for example, the Arduino Nano (figure 5), it uses more power and it takes longer to compile programs for.

Figure 6 - ESP32

Controlling 12 motors at once

Choosing a motor type

There are several different motor types to choose from when building a robot, namely; DC Motors, Stepper motors and servos. For our robot we have several requirements:

The motors must be

- Small
- Controllable by Arduino
- Low in power consumption
- Capable of precise control

DC Motors cannot be controlled precisely and are therefore not an option.

Figure 7 - Stepper motor with separate control circuit

Stepper motors are very precise but they are big and, most importantly, they require a separate control circuit per motor as seen in figure 7 and therefore are not suitable for this project (Elprocus, n.d.).

Servo motors are therefore the best option for this project. These motors use a built-in SG90 controller IC which can be controlled by PWM over a one-wire interface and allows for 180 degrees of rotation. Both the servo-axle and the supplied arms have teeth that grip onto each other to prevent slipping. It is important to note that servos can only hold their position when power is applied to them.

Figure 8 - The SG90 micro servo motor used

Pulse-width modulation

The servo motors are controlled by a pulse-width modulated signal at 5 volts DC. Pulse-width modulation (PWM) is used to send analogue values over a digital data line, it does so by alternating the width of a constant pulse rather than changing the voltage like an analogue data line. A PWM signal runs at 50 Hz and therefore the maximum width of a pulse is 20 ms, the minimum is 1 ms. The width of the pulse is called the duty cycle (Barr, 2001).

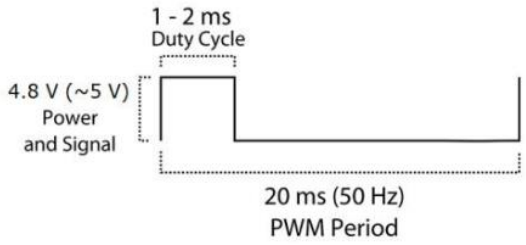


Figure 9 - One PWM period of the servo motor

According to the datasheet of the SG90 controller (figure 9), these servo's use the duty cycle between 1 ms and 2 ms, thereby dividing the 180 of rotation over 1 millisecond. The duty cycle is measured by the SG90 controller and turned into an analogue value which turns the motor to the specific angle value. Values outside of the 1 – 2 ms range cause undocumented behaviour (SERVO MOTOR SG90 DATA SHEET, n.d.).

Controlling

An issue raised by using PWM controlled servo's is that the microcontroller we are using is not capable of generating 12 separate PWM signals, because the Arduino does not have enough pins that can generate PWM signals. For this reason, a separate controller board is required. One such board is the PCA9685 (figure 10), this is a 16-channel PWM generator IC which is generally used for controlling LED and servo's. It allows controlling 16 servo motors at once over the I^2C bus and therefore only needs two data lines to the microcontroller (NXP Semiconductors, 2015). The I^2C-bus protocol is a way of communication between Integrated Circuits (IC) (NXP Semiconductors, 2014).

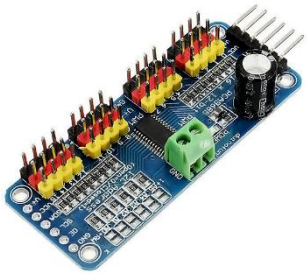


Figure 10 - PCA9685

Coordination

A robot needs a way of positioning itself so to prevent damage to itself or anything else in that regard. There are several ways to allow a robot to position itself such as GPS or through a connection with several base stations in its moving environment which report proximity back to the robot. The moving environment of the robot is predefined and the robot determines where it can and cannot move based on the information it receives from the GPS Satellites or its base stations. Such as system is commonly used in automatic lawnmowers.

Figure 11 - HC-SR04 ultrasonic sensor

GPS or base station based position is very precise and a robot can predetermine its pattern of movement beforehand knowing the borders of its environment. Another system, which is more commonly seen in robots of the nature we are building, is more direct as it works with proximity data. Using LiDAR or ultrasound, a sensor can determine the proximity of objects in a short range of itself and the robot can change its pattern of movement based on that information. LiDAR and ultrasonic proximity measuring work similarly. LiDAR uses a pulsed laser signal and measures the time between sending and receiving that signal. Using the formula $s = c * t$ the sensor can determine the distance to the object. Ultrasonic measuring works using the same principle but instead uses ultrasonic sound waves

(Sharma, 2020). There are sensors available that work with Arduino for both LiDAR and ultrasonic based proximity measuring. One such sensor is the HC-SR04 (figure 11) which has two separate transmitters? One for sending and one for receiving ultrasonic waves. According to its datasheet, this sensor has a maximum range of about 4 to 5 metres over an angle of 30 degrees (ITead Studio, 2010).

Writing software for the Arduino

Arduinos are programmed in a specialized version of the C++ programming language (Arduino, n.d.). C++ is a general-purpose programming language that can be used to program high-performant programs (games for example) or for systems that do not have many computational resources (like Arduinos). C++ is a mature language, in that the language is completely standardized (ISO/IEC, 2013), changes to the language are carefully considered by many international parties. The C++ code is, firstly, compiled; that is to say, the human-readable C++ code is "translated" into machine code that is easily executable for the computer (cplusplus.com, n.d.). Then, the compiled code is sent to the Arduino board and it executes the program. The Arduino can only run one program at the time.

The Arduino's specialized version of C++ includes the same basic functionality as regular C++ such as if-statements and while-loops, besides that it also includes functions for interacting with the pins at the sides of the Arduino board. These pins can be switch high (3.3 volts) or low (0 volts) or emit a PWM signal programmatically. The Arduino also has functions for transmitting data from the board to the computer and vice versa over a Serial to USB interface. These functions are helpful when it comes to debugging a program.

Designing and producing the parts

Designing

The parts for the robot will be largely based on measurements from the model, the length between two joints can be determined using vector math. Using that data, simple legs can be designed using a modelling program such as Sketchup or AutoCAD.

The legs are fairly simple, the axle-to-axle length will be equal to the joint-to-joint length derived from the model. As the cat has two joints per leg, two motors will have to be attached to the leg itself. The servo motors have two attachment brackets on the short side, therefore a square hole will be made into the legs that have to support a motor.

Material

Choosing the right material when building a robot is essential, the material must be strong but should not be heavy for the robot to function correctly. The most notable materials to choose from when building a robot are metal, wood and plastic

Lightweight metals such as aluminium are great for building robots as they are easy to make without specialised tools, however, this is also a problem as lightweight metals are subject to bending both during production and operation which can be a serious problem. While this can be prevented by creating right angles in the metal, doing this manually is generally not very precise and this makes editing the material less practical. Another option is to use a T or H style metal profile, however, this is generally heavier and more expensive. Next to that, creating metal parts properly, without any sharp edges, is quite difficult without specialized tools.

Another possibility is to use PLA plastic produced by a 3D printer. 3D printing is an easy way to put 3D models into reality as most modelling programs allows for exporting to a 3D printable format such as stl. When using a good 3D printer, the results are precise and do not require any post-production work like painting or sanding as you would need to do with metal or wood. PLA plastic is very strong and not flexible at room temperature

The last option is to use wood. Compared to metal and plastic, wooden parts are easy to create precisely without special tools, lightweight and easy to connect. To make parts precisely, one could use a CNC machine. The CNC machine drills part out of a sheet of wood, based on a 2D vector model. Wood does require sanding and filing to prevent splinters.

Method

Acquiring data

To create a model we plan to use a regular video of the cat with markers walking a straight line. CMA Coach 7 will be used to process the data as that is what is available for us to use via our school. The resulting model in coach will be exported as raw data to be used in the robot.

Putting the Data into movement

We'll be using an ESP32 dev board as it has more memory and processing power which will be required to store and process the model data. To circumvent the longer compiling times, we used a Linux based development environment as compiling any C based language on Linux is generally more efficient.

Controlling 12 motors at once

Choosing a motor type

We chose to use servo motors for our robot. We had two different models to choose from, metal and plastic geared motors. To determine the strength of these different models we conducted a short experiment as follows.

The motors' strength was tested by connecting a right angle arm piece to the axle, connecting 50-gram weights with a loop of fishing line and increasing the weight until it was no longer able to lift the weight 90 degrees. The results were surprising, the metal-geared servos were able to lift 300 grams before they started to struggle. The plastic geared motors on the other hand were able to lift up to 500 grams with no issues. We stopped measuring at 500 grams as we were afraid that the servos might have broken if we increased the weight further and it is unlikely that the servos will be subjected to higher amounts of force.

Coordination

Due to the robots simple nature, we will be using a single HC-SR04 pointing toward the front of the robot so it can prevent itself from walking into things.

Designing and producing the parts

Designing

To design the parts, we will use AutoCAD as it supports various ways to design and export 2D vector models precisely.

Material

At first, 3D printed plastic seemed to be the best option, however, the 3D printer we have at our disposal, the Anet A8, is not very precise and therefore the result will require sanding and filing to make the parts fit together. Next to that, after printing some test parts, we found out that combining PLA parts is more complicated compared to wood or metal as screwing into PLA is nearly impossible without causing it to break. And if the parts break, it takes nearly 1.5 hours to reproduce them. For these reasons, we have decided not to use 3D printed parts and go with wooden parts instead.

We do have a CNC machine available for us to use, therefore we will use that to create wooden parts. The CNC machine in question is a modified Brink KM3 bench sized engraving machine.

Assembling the robot

To assemble the robot we plan to connect the arms provided with the servo motors to the wooden legs using short screws. To connect to motors to the wood, we will be using a nut and bolt as the motor will have to be connected to the wood securely and must not be able to move while in operation.

Summary

Recording the cat's walking pattern

The cat was recorded walking a small distance over a bench using a smartphone (figure 12). A measuring tape at one meter of distance between two bottles (not associated with the experiment) was put on the table in the background to allow Coach to measure at the correct scale later on.

Figure 12 - A frame from the recording of the cat

To get the cat to do what we wanted we lured him using some food.

We used the footage of one walking cycle which was about 22 frames long.

Turning the video into workable data

Coach 7

The video was loaded into Coach 7 using the template 'Lege videomeetactiviteit'. Coach 7 would normally be able to track points automatically and plot the values on a diagram. However, because the colour of the cat's fur does not stand out enough from the background and because the colour of the cat's fur is nearly even over his entire body, Coach was not able to track the joints automatically.

An attempt was made to film the cat again using buttons as markers to make this process easier, however, the cat did not want to cooperate and refused to take a step without flicking his legs which caused the buttons to either fly off or become displaced.

In the end, the measurements were done by hand.

The measurement data consists of 16 sets of x, y coordinates, 8 for each side, which represent the joints of the cat. These values were very hard to read and therefore we created another program the visualise the data: catmesh_pws.

Catmesh_pws

Catmesh_pws is a program that we have created to model the data from Coach 7. We have done this to create a better overview of what we needed to do to make the final product. Catmesh_pws contains a visualizer for the raw data from Coach 7 and tools to export the data in a fashion which is readable for the Arduino.

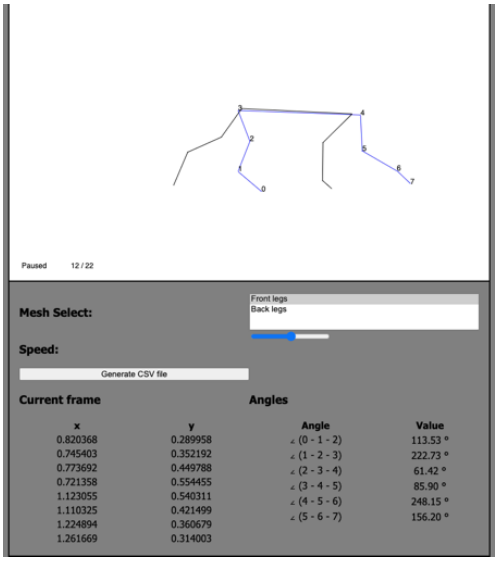


Figure 13 - catmesh_pws

Catmesh_pws is built using *Electron* and *asdf-games*. These are programming tools called frameworks. Frameworks are sets of pre-programmed code that are ready to be used by any programmer. *Electron* handles the creation of the window the application starts in, and *asdf-games* handles the logic behind the visualizer. We coded in the JavaScript programming language. We chose for this programming language because we have a lot of experience programming in JavaScript and we knew some tools that can help us with that. Moreover, JavaScript is

not suitable for bigger programs that use a lot of computational power. That is not a problem for catmesh_pws, however, because the program is not big enough to cause those problems.

A principal feature of catmesh_pws is the smooth animation of the cat's "bones". This is made possible due to linear interpolation. Linear interpolation is a method to add more data points to a given range of data. In catmesh_pws it is implemented by calculating the progress of the frame (in figure 14 called *lerpProgress*), that is, for how long the frame is drawn relative to a maximal amount of time. Then, multiplying the target of the line given by the original data points from Coach 7 with this progress variable and adding it up to the origin of the line, gives the interpolated coordinates for the bone. This calculation is made approximately 60 times per second when the animation is running. The progress of the frame is dependent on the maximal amount of time, which can be controlled using the slider underneath the wireframe cat. The figure below shows the code for the linear interpolation implemented in catmesh_pws.

```
// Calculate the coordinates for the line. If we're not at the last frame, use linear interpolation to make a smooth animation.
let originX, originY, targetX, targetY;
if (frame < this.frames.length - 1) {
  const nextFramePoint = this.frames[frame + 1][index];
  const nextFrameTarget = this.frames[frame + 1][index + 1];

  const lerpProgress = (this.redraw.max - this.redraw.current) / this.redraw.max;

  originX = (point.x * this.scale) + (nextFramePoint.x * this.scale - point.x * this.scale) * lerpProgress;
  originY = (point.y * this.scale) + (nextFramePoint.y * this.scale - point.y * this.scale) * lerpProgress;
  targetX = ((target.x - point.x) * this.scale) + ((nextFrameTarget.x - nextFramePoint.x) * this.scale - (target.x - point.x) * this.scale) * lerpProgress;
  targetY = ((target.y - point.y) * this.scale) + ((nextFrameTarget.y - nextFramePoint.y) * this.scale - (target.y - point.y) * this.scale) * lerpProgress;
} else {
  originX = point.x * this.scale;
  originY = point.y * this.scale;
  targetX = (target.x - point.x) * this.scale;
  targetY = (target.y - point.y) * this.scale;
}
```

Figure 14 - The code for linear interpolation in catmesh_pws

Another important feature is the way data is handled by the program. The program imports raw data provided by Coach 7. Coach 7 provides its data in the CSV format. CSV files are structured similarly to how tables are structured. The file starts with a header, which essentially describes the data that is inside of that column. After the headers, the data is saved in columns, delimited by a fixed character (frequently a semicolon or comma), and every line represents a row in the table. The first line is special, as it represents the header of each column; a name describing the contents of that column.

The data exported by Coach 7 has 17 columns, one for the time – which is irrelevant for catmesh_pws – and 16 columns for the x and y values of 8 joints. Moreover, the file holds 22 rows, because the recorded video had 22 frames.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12
119.50	222.30	110.54	69.97	215.52	203.27	102.51	255.50	36.47	74.50	210.04	186.68
135.00	200.33	124.04	62.88	220.79	198.52	102.03	260.63	36.79	87.74	236.34	122.40
128.95	212.89	120.13	61.26	211.07	204.66	105.46	245.62	50.10	113.43	223.00	163.35
117.04	237.26	97.11	57.90	211.93	203.78	99.97	248.69	59.40	124.00	198.11	177.89
109.54	251.09	82.04	49.54	217.18	217.02	106.76	241.35	66.97	132.15	179.82	205.49
99.55	263.46	58.15	49.90	208.18	205.42	98.96	234.96	87.01	120.00	194.37	205.15
95.55	258.44	53.41	45.71	204.70	202.75	97.78	234.37	92.43	111.18	194.17	221.19
107.55	236.70	51.04	51.64	210.53	137.50	118.20	222.50	95.87	103.58	191.23	228.72
122.73	213.08	52.34	48.69	233.96	123.05	105.57	219.12	110.56	107.24	181.18	225.46
120.79	224.21	48.74	46.88	252.67	118.99	123.08	210.23	123.08	82.68	205.54	231.76
120.08	224.21	53.72	65.40	259.04	129.55	133.40	195.27	129.93	61.30	225.01	214.98
113.53	222.73	61.42	85.90	248.15	156.20	141.53	206.31	124.88	41.84	233.31	223.13
104.79	220.48	73.50	103.97	226.01	176.16	132.72	214.79	119.99	51.93	216.72	236.58
104.68	235.25	68.46	123.84	189.58	197.07	118.07	241.44	97.65	33.47	228.93	244.88
102.67	240.48	71.69	114.80	192.66	223.48	109.56	260.87	67.83	37.39	222.76	232.82
111.18	219.74	87.41	102.48	213.53	207.96	107.87	261.72	51.80	41.10	211.71	229.57
120.96	219.29	89.70	103.60	203.37	212.47	91.41	261.85	45.10	36.92	208.53	215.36
107.88	231.19	89.16	92.53	205.64	229.42	100.45	263.48	27.04	29.76	236.72	133.64
105.81	242.63	84.27	83.02	214.67	217.34	121.26	230.69	37.61	37.47	246.71	118.52

Figure 15 - The csv file read by the Arduino

The data that is exported by catmesh_pws (figure 15) and used by the Arduino has 12 columns, one for each rotatable joint, that contain the corresponding motor's rotation for each frame in degrees. This data will later be converted to workable data for the servo motors by the Arduino.

To calculate the proportions of the robot components, the length of the vertices needed to be known. The video supplied in Coach had a reference scale that Coach could use to determine the x and y coordinates. The table exported from Coach contains

```
var vector1 = {
    x: lines[index + 1].x - lines[index + 0].x,
    y: lines[index + 1].y - lines[index + 0].y
}

var vlength = Math.sqrt(
    Math.pow(vector1.x, 2) + Math.pow(vector1.y, 2)
);
```

Figure 16 - Code accounting for the length of the vertices

the x and y coordinates of the joints in meters. The distance between two points in the table is equal to the distance between the corresponding joints of the real-life cat. The distance between two points can be calculated using the Pythagoras theorem. The code in figure 16 shows that, firstly, the difference in x and y is calculated (lines 1 -> 4) and the result is used in the Pythagoras theorem (lines 6 -> 8).

Creating the Arduino circuit

Hardware

The components of the circuit were simply but on breadboards using jumper wires to connect the various parts using the following schematic (figure 17).

The SPI and I²C bus pins on the ESP32 are predefined, therefore only those specific pins can be used. Other types of data connections such as the button, led and proximity sensor can be established using any of the digital input/output pins.

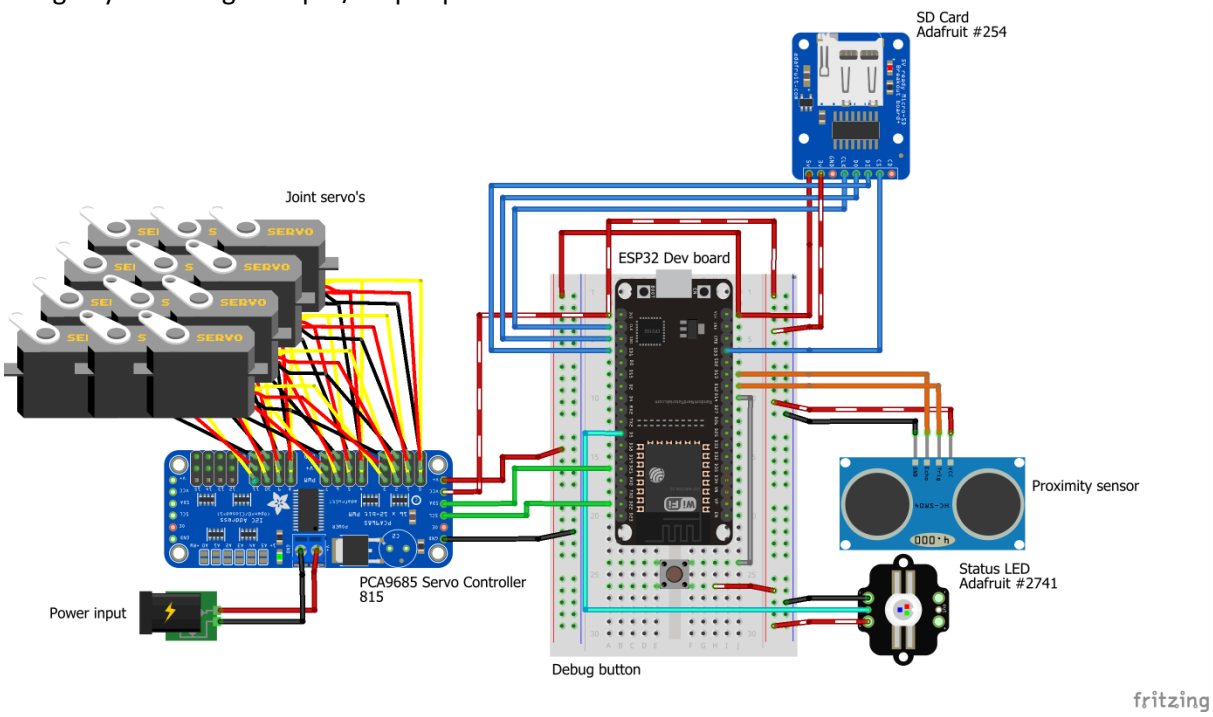


Figure 17 - Schematic showing the different parts of the circuit

The wires colours are as follows:

- Ground — Black
- 5 volts — Red
- 3.3 volts — Red / White
- I²C Bus — Green
- SPI Bus — Blue
- RGB data — Cyan
- Proximity data — Orange
- Servo PWM — Yellow

The connections are as follows

- All components have a common ground.
- 5 volts power is connected to the power input of the PCA9685

- PCA9685 Servo Controller is connected to the ESP32 via the I^2C bus (SCL, SDA), 3.3 volts and 5 volts which powers the ESP32
- The 12 servos are connected to the PCA9685 to ports 0 to 11 respectively
- SD Card is connected to 5 volts, 3.3 volts and the SPI bus (CS, SDO, SDI and SCK).
- The Status LED is connected to 3.3 volts and digital pin 5 for data input.
- The Debug button is connected to 3.3 and digital pin 14, hence D14 is pulled high when pressed.
- The proximity sensor is connected to 3.3 volts VCC and digital pins 12 and 13 for data.

Storing Model data

The data from the model turned out to quite big, therefore programming it into the memory of the ESP32 was both very slow and unpractical. To remedy this we put the CSV data from catmesh-pws on a Micro SD card and read the data of the SD card using the Arduino. While this does cause the initial start-up routine to take longer, it is quicker to debug and it makes it easier to replace and edit the model afterwards.

To connect the SD card to the Arduino we will be using a Micro SD card breakout board which converts the 5-volt logic used by the Arduino to a lower level that can be used by the SD card. The Arduino will communicate with the SD card over the SPI bus. Theoretically, one could connect the SD card to the Arduino without any conversion circuitry, instead using a series of resistors. However, using a specialised IC (integrated circuit) like the one on the breakout board could prevent damage over long term use.

Power usage

By putting an amperage meter in series with the main power input we determined that the average power usage is about 200 mA during normal operation and 400 mA max when all motors are rotating.

Software

The circuit consists of two parts: the Arduino and the servo controller board. The Arduino controls the servo controller board and the servo controller board controls the motors. The communication from the Arduino to the servo controller board is specific for the type of board, therefore the manufacturers of the controller board have created a code library for interfacing with the controller board from an Arduino. The code library can be used to programmatically control the servo controller board and, thus, control the motors themselves.

Loading the model data from the Micro SD card into the Arduino's memory requires memory allocation, this way the compiler deliberately leaves space in SRAM so the data can be loaded in by the Arduino itself during the boot-up sequence. The data table, however, was still too big. The Arduino physically had enough SRAM, however, loading the data simply took too long. Next to that, the Arduino needed to calculate the angles for the servo motors for every frame over again which made the movement not run very smoothly. We did try to make the Arduino load and process the model data into usable angles during the start-up routine, however, due to the limited processing power and memory this caused the robot to take nearly a minute to boot which was not very convenient. For this reason, we decided to shift these calculations over to catmesh_pws.

The data exported by catmesh_pws is optimized for the Arduino to use. The data represents a table with angle values for each motor at every frame of the original video. The Arduino linearly interpolates between the values to create a smooth movement. We have used open-source libraries to correctly read and parse the data. Firstly, we used the built-in "SD" Arduino library, which provides functions

for reading the SD card. Secondly, we used the open-source CSV-Parser-for-Arduino (available on GitHub.com) library, which provides a function for parsing the angle values stored on the SD card.

Once we have the angle data ready, we can send it to the controller board. To allow our motors to be controlled using the angle values, we, first, had to figure out the minimum and maximum duty cycles. The values stated in the SG90 controller datasheet could not be used as-is because we noticed the values were off when testing if the motors were functional using a servo tester. After some investigation using a portable oscilloscope, we found out that the servo tester was indeed using the correct duty cycles, yet the motor did not rotate a full 180 degrees. For this reason, we decided to find these values ourselves.

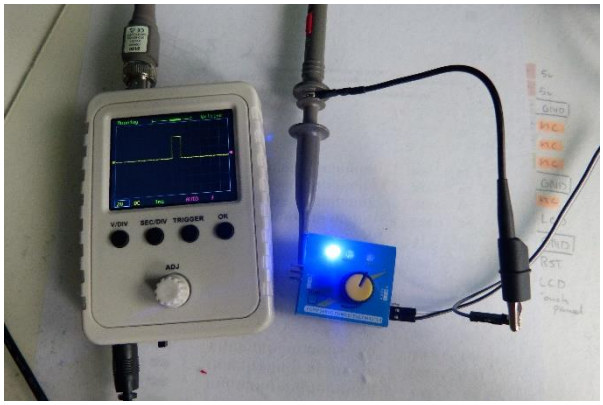


Figure 18 - Oscilloscope used for investigation

The library associated with the PCA9685 uses ticks to control the position of the servo, we created a simple Arduino sketch that allowed us to enter tick values manually through the Arduino Serial monitor. After that, we tested various values trial and error until we got the tick values which corresponded to 0 and 180-degrees of rotation. By putting these values in the following formula, the tick value of any angle between 0 and 180 can be calculated.

$$\frac{angle}{180} * (MAX_{Tick} - MIN_{Tick}) + MIN_{Tick}$$

The code library uses ticks to divide the PWM period into segments. The PWM period is 20ms and divided in 4096 segments (12 bits; 2^{12} = 4096) (Adafruit, 2020). So, the number of ticks we found for a rotation of 0 degrees (MIN_{Tick}) is 130 and the number for a rotation of 180 degrees (MAX_{Tick}) is 560 when a frequency of 50 Hz is used. When the frequency changes the number of ticks needs to change too.

To receive data from the proximity sensor we used a small library instead of the official one to preserve space, this library was created by a Danish man, hence its name Afstandssensor.h. This library is very basic as it only has a single function; `afstandCM()`. This function returns the distance between the sensor and the nearest object in centimetres. If the sensor reports anything under 10 centimetres, the robot will stop moving.

Creating the parts

At first, the plan was to model the parts in AutoCAD and then convert them to a format supported by the CNC machine, however, the software used by the CNC machine, KM3 Wincam, is very old and did not support any modern format. Converting the files from AutoCAD seemed to be impossible, therefore we designed the parts in Sketchup, which allowed exporting the model to format supported by KM3 Wincam. Another problem arose, KM3 Wincam is very basic and does not account for the width of the drill bit used and therefore the resulting parts will be too small and have holes that are 3mm bigger than they should be. All of the measurements taken in Sketchup would have to be altered to account for the width of the bit manually which would take a lot of time. Next to that, after several test runs, we concluded that the CNC machine was very inaccurate and the speed of the milling had

to be quite slow to prevent the wood from breaking. We decided it would be quicker and easier to saw out the parts by hand using a table saw.

The legs for the first prototype were made using a table saw at school. The Sketchup model was exported 1:1 and printed out on paper, the sheet was then glued onto a sheet of triplex wood and sawed out. The holes for the motors were first drilled out, then sawed to shape using a table saw and, if necessary, filed to account for inaccuracies.

The most principal part of the robot is the main body. It is a wooden structure shaped like an "u". The cat's legs are attached to the sides of the main body. In the middle of the main body is abundant space for the circuit. The main body's construction is very simple. We sawed three planks – two of equal width and one slightly wider – and we screwed the two planks of equal width perpendicular to the third plank.

Figure 19 - The main body of the robot

Wood choice

The first prototype of the main body used teakwood because that was readily available to us. We later found out that teakwood was too heavy and we switched to another type of wood: triplex.

Our first choice for the type of wood to use for the legs was triplex as it is light, strong and thin. This turned out to be a mistake as triplex wood turned out to be very brittle when sawed in small pieces as we did. The cause for this is because triplex wood is made out of three, thinner sheets of wood glued perpendicular to each other, creating a crossing pattern. When the parts were sawed out, some parts of the outer layers were only half a centimetre in size causing them to break off. (figure 20)

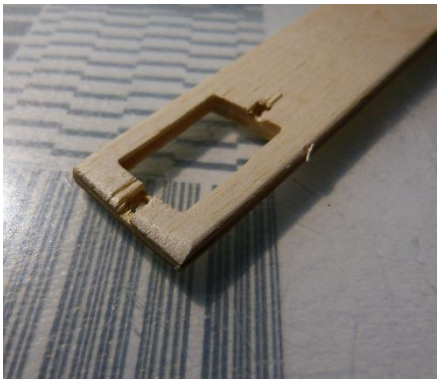


Figure 20 - Leg component with splintered wood

Assembling the robot

Integrating the servos

Once all parts had been manufactured, the robot had to be assembled. The first step was to label the wooden pieces to their corresponding leg part from *catmesh_pws*. This could be figured out by comparing the lengths of the average values calculated in *catmesh_pws* to the lengths of the wood pieces. After we could tell the pieces apart, we needed to connect the servos to the pieces with holes reserved for them. We did this by simply screwing them in place (figure 21). The same had to be done with the U-shaped shell that represents the cat's main body.

Figure 21 - Leg component with motor installed

While the servos themselves were in place, it could not move anything because nothing was connected to it. That is why we need the servo-arms. These are attached to the wood and servo with

Figure 22 - Close-up of the connection between motor and wood

screws (figure 22). For the screws to fit, the gaps in the servo-arms needed to be widened with a drill. It is also important that the wood, and thus the servos, face the correct way. If all servos faced the same way, the legs would bend outwards and would likely cause the robot to break and fall apart. If the servos alternate between facing forward and facing backwards the robot is much more stable as the mass is more evenly distributed.

First prototype and testing

Figure 23 - The first prototype

In figure 23, you can see the first prototype. The version depicted does not have the Arduino and the rest of the circuit in place which would be connected to the wires. These were also the first problem of this prototype. The wires were way too short which would restrict the movement of the cat greatly. Furthermore, the teakwood that was used for the U-shaped shell, which was deemed too heavy. Before trying to let it walk, we first put it on a stand (as depicted) and let it run. This failed, as it resulted in the legs flailing around. Because the wires were so short and tightly connected, there was nearly no movement space and everything broke down.

Even though this did not go as planned, we did discover a mistake in the Arduino code: the servos were moving for an extra, non-existent frame, making the robot run for 23 frames instead of 22. After some inspecting, we found that there was one < that had to be replaced by <= (smaller than or equal to). Now the code did work, but the shell had to be replaced with a lighter wood type and the legs had to be shortened.

Second prototype

Resizing the parts

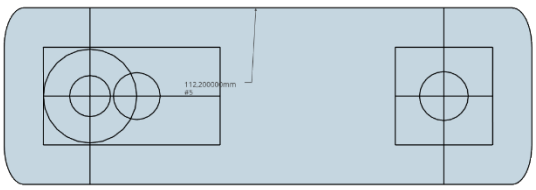


Figure 24 - Sketchup template of a resized leg component

Creating the correct size was a matter of trial and error, so we went for 50% the original size and 75% the original size. This was done in SketchUp on a duplicate of the old file. Resizing meant that all the lines indicating the placement of servos and servo-arms had to be removed beforehand. This was because the lines had to be exactly 5 millimetres removed from the edge. This was made more difficult by the fact that SketchUp does not snap the to be moved object

to the most proximate rounded position; in other words, the placement of the cursor had to be exact. Another problem that we encountered, was that SketchUp binds the version number of the SketchUp used to the file. Because we used SketchUp 2016 and 2020, the file was not compatible. It was created in 2016 and could be opened in 2020 but not in any older version. Even though there was an option to export the file in an older format, for SketchUp 2017. This also did not yield success. Therefore, we had to use the third version of SketchUp (2017) to convert that file back into 2016, which it was capable of. We decided to make the 75% version a reality as the other was deemed too small.

Producing parts

The parts for the second prototype were produced quite differently. First of all, another type of wood was chosen. Pinewood profiles were used instead of triplex wood, not only was this wood lighter, it was already cut into strips of the correct width. However, this wood was even more annoying to saw and drill into as it broke very easily when punctured. We were able to produce all parts successfully eventually by wrapping the wood in construction tape which prevented it from breaking.

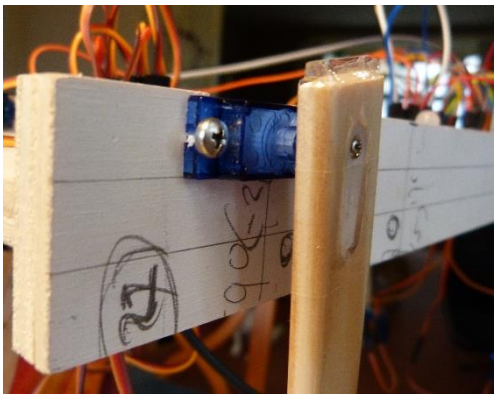


Figure 25 - Close-up of the prototype using the new wood

Another technique was used to connect the legs to the servo motors, as seen in figure 23 the distance between the servo and the leg on the first prototype was too big which made the legs flimsy. In order to lessen this distance the connecting arm of the servo was put on the opposite side of the wood as seen in figure 25.

All of the parts were fixed to the chassis using screws instead of double sided tape.

Changes to the circuit

The robot was still too big for the motor wires to reach the servo controller, therefore another one was added to the circuit. To prevent having to change the code significantly, the second controller was connected parallel to the first so that they would both respond to a single command. Some of the wires were still too short however, these were extended using wires salvaged from other servo's that broke during the testing and building phase of the first prototype.

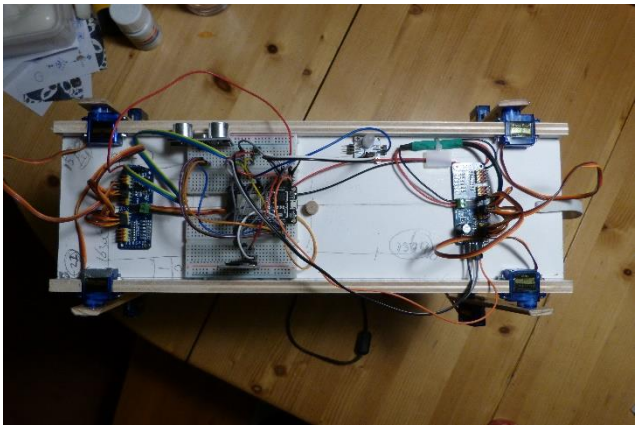


Figure 26 - The new circuit

Changes to the code

Using the model with the angle values still seemed to be an issue. Some of the values in the model were over 180 degrees of rotation and we accounted for that in the code by simply taking the angle and deducting 180. This caused some problems however as some of the values to be lower than zero which would then cause the motors to malfunction. To fix this we changed the way the motors are controlled. Instead of sending the angle from the model to the motor directly, the Arduino calculates the difference between the current step and the next and then sends this to the controller. Every motor has a base value of 90 degrees which is equal to the first frame of the walking sequence.

Because the joints in the model never turn more than 90 degrees, the angle of the motor will never exceed 180 or become less than 0. An example of the control sequence is described below.

- Frame 0 does not change the position of the motors, this is the base value on which the robot is assembled
    - Every motor is at a physical angle of 90 degrees, thus it can move 90 degrees to either side.
- Frame 1 calls for a change of -5 (e.g. 187 -> 182) degrees for motor 3
    - This change is divided over 50 steps to make the movement run smoothly
    - Every step between frame 0 and 1 calls for a physical change of -0.1 degrees
    - On frame 1, the physical angle of motor 3 is at 85 degrees

Results

Cat robot

The cat robot is the final product for this *Profielwerkstuk*. The goal was to make the robot able to walk by itself and (if we had time to spare) to make the robot recognize walls and other obstacles and navigate around them. Unfortunately, we were not able to make the cat manoeuvre around obstacles because of a lack of time. The cat is able to make rough movements. These movements, however, do not allow the robot to move itself. The servo motors do not make the right movements, and thus, they do not make a proper walking cycle.

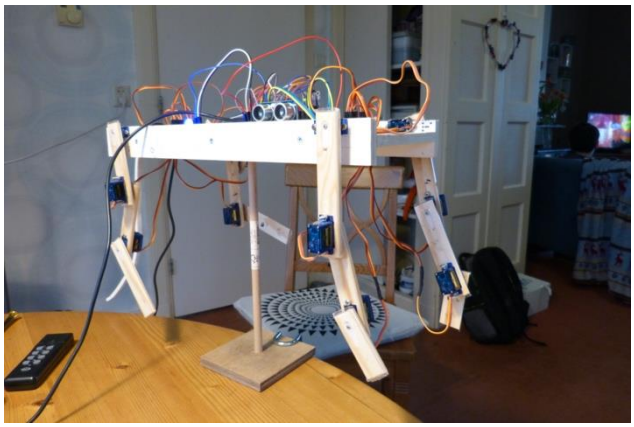


Figure 27 - The final product

The robot has four legs, each one having two joints in the legs and one joint to connect to the main body. Each joint has a servo motor that allows the associated leg part to rotate. The main body holds all the circuits and connects all the legs together. The circuit features an esp-32 (a type Arduino), a sonic distance sensor and two servo controller boards.

Other features that the robot has are:

- a status light that indicates when an error has occurred and what type of error it is. It shows this by blinking a red light a specific number of times. If the light blinks twice, there is an error reading the SD card.
- an SD card reader, this is used to easily swap the model data for the robot. The Arduino searches for a file called “data.csv” and reads that to turn the data into movements.

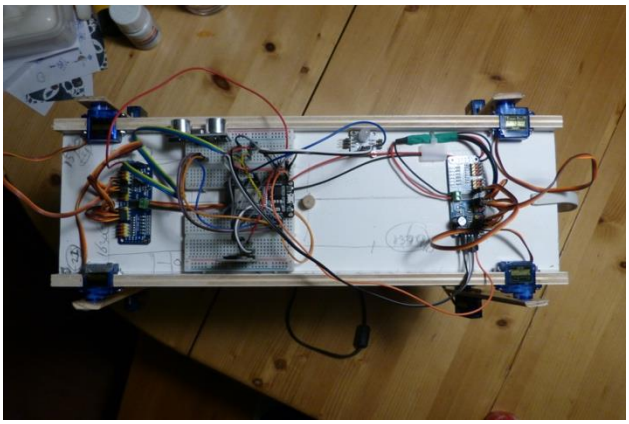


Figure 28 - The final circuit

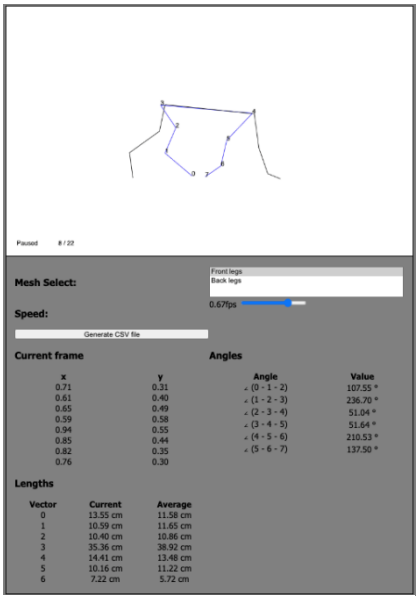


Figure 29 - The final version of catmesh_pws

Catmesh_pws

Catmesh_pws is the visualisation program we created. The program’s functions are:

- Visualising the data imported from Coach. Coach supplies data for the points’ X and Y positions. This data is displayed on the canvas and all the points are numbered and connect to the next point. Pressing spacebar toggles the animation: the points start moving to the position as it can be seen in the subsequent frame of the original cat recording. The animation is created by a technique called linear interpolation. The animation can be slowed down or speeded up using the “speed” slider.
- Transforming the imported data from Coach into a readable format for the Arduino. The X and Y values from the Coach file are converted into angles (in degrees), which the Arduino can use to control the servo motors.
- Calculating the average lengths of lines between two points. This is used to determine the length of the parts for the robot’s legs.

Conclusion

We entered this project with a rough idea of what we wanted and how we could accomplish the construction of the robot. To start with, we needed to record a cat's movements and turn the footage into workable data. This was possible due to the Coach program provided by school. In order to aid the transfer of data to the Arduino and its motors, we created the catmesh_pws application. This program has helped us a lot and we kept updating the program throughout the research phase. After some fundamental features were in place, such as more control options and linear interpolation, it was time to start working on the Arduino code.

We first needed to research how pulse-width modulation (PWM) worked and subsequently how to convert the angles generated in catmesh_pws to PWM frequencies. It was found however, that although we verified that the values listed on the datasheet for our motors were correct with an oscilloscope, they didn't exactly turn as much as they should. Therefore we had to come up with our own values.

The next task was to produce the wooden components of the cat. We designed the leg components using SketchUp, contradicting our initial plan to use AutoCAD. After the leg components were printed and sawed and the main body was produced, the cat's assembly started, which made us discover a bug in the software. The next problem we encountered was the fact that the legs were too big for the wires to connect the motors on them. Even after resizing the legs, it was still not enough. We had to install another breadboard on the cat's body so that we could divide the wires for the front and back legs.

While the reorganisation was a success, the motors still did not do what we wanted. At this point, we had checked the code many times and while we were still finding issues, none of them fixed the weird movements the cat made. It was until the very last meeting that we made some progress in the way the cat moved. Sadly, it still wasn't close enough to the catmesh_pws model.

In the end, creating and planning the construction of a robot is a very complex task, and presents those who attempt with many hurdles along their way. While it seemed feasible with the limited tools we had and we still managed to come close to our goal, the robot did not turn out as we would have liked. We know it is possible the way we did it, yet either bugs in the software or faulty hardware caused the project to fail.

Discussion

Our research is not really trust-worthy. That is because the goal of this research was to research how to build a robot. We think that we have documented our process enough for someone to be able to replicate the steps we also did. However, it would be hard for someone to replicate researching how to build a robot. The manner we did everything is one of many ways one could have done it.

Overall, the research went well. The cooperation between the group members was close to superb. However, the communication with the tutor was not great. It was sometimes a bit slow, but we eventually received answers. Moreover, our tutor gave us useful feedback on previous versions of reports.

In terms of the execution of this research, there are several points we could have done better. One major issue was the planning. We had not planned ahead of time enough, which led to some stress and a slightly unfinished product. If we had planned more ahead, we could have been able to have a more finished product. We did quite a lot of work after handing in the first version of this report, which was, reporting back on it, a bit too late. At the start of the project, we had faith that we could finish it in time, and we even thought we were not going to fulfill the 80-hour goal before finishing the robot.

The final product sadly does not work as expected, we think this is because the motors we used are not accurate enough to reproduce the movements in correct fashion. We could have solved this by decreasing the step size between frames, however this would have made the movement quite violent which would still render the robot unable to walk by itself. Another way we could have solved this is by creating a model that has more frames in the first place. The video of the cat was only 22 frames long and therefore the difference between the frames was quite large. If we had recorded the video at a higher framerate the Arduino would have to do less calculations to smoothen the motion. However, this could bring about an issue regarding the size of the model and Arduino's memory capacities.

Besides, the motor's axels were not strong enough to hold the weight of the robot, this is another reason why the finished product is not able to walk. The motors are simply too small. And, making the robot itself smaller would have made it impossible to fit all the components in at the current state.

Because the cat was not completed as planned, some features were left out, for example, the distance sensor. Although we planned the distance sensor for in case we had time left.

Personal reflection

Arne

During this PWS I learned that I greatly underestimated the work it takes to build a functioning robot from scratch, already knowing a couple of things about arduinos and woodworking I thought it would be a piece of cake. It was not, we ran into several issues that we didn't expect.

The research part went great in my opinion, tasks were divided equally and everyone did their part without question. I enjoyed doing the experiments before starting working on the main thing such as testing the motors which also went well in my opinion. Everyone did their part in this PWS and the overall cooperation went very well.

There are issues that I should have expected beforehand such as the signal interference on breadboards which caused the circuit to run instable. We were able to fix this afterwards by soldering most of the important connections. However, the motors not being able to support the robot was quite a surprise to me, maybe this is just because of our build or the motors themselves as I have seen people do heavier projects with these things.

Nevertheless, it became very obvious that our robot was not going to walk anywhere anytime soon, the entire thing was simply too bulky and to flimsy to even stand on its own. Overall I am quite disappointed that we were not able to produce a working result, but I did learn more things about modelling using Coach, woodworking and communication protocols like PWM and I^2C. If we were to give this project more time after handing in this document, I'm sure we would be able to make a function robot, however, making it accurate to a real cat with this setup seems to be impossible and that was the point of this project.

I am happy with the feedback our tutor was able to provide as we made his first PWS quite hard. A tip I would like to give to those to do PWS next year is to make a planning and stick to it. Next to that make sure that your entire group is interested in the subject, reading the other's reflections I realise that we probably should have thought longer about the subject of this PWS.

Job

This PWS has taught me a lot. Building a robot, showed me how to work with more advanced circuits, how to work with servo motors and how to integrate hardware components into a wooden structure. One of major problems that I encountered was my lack of knowledge about woodworking. The robot required a fair amount of woodworking skills that I did not have. Luckily, Arne was on our team and he knows a lot about it.

My expertise was more on the programming side. I contributed largely on catmesh_pws and I am really proud of it. The program turned out to be quite helpful when it came to assembling the robot. Moreover, catmesh_pws is one of most stable and perfectionated parts of this PWS.

Overall, I am pretty happy with how the PWS has come about. Even though we did not end up with a fully complete and competent robot, I think we have captured a pretty good essence of how one could go about making a robot of this nature. And with a better planning, things would have gone much smoother and it is a shame that our planning was not optimal, because I would have loved to finish up the robot in time. This working with servo motors and Arduinos has made me excited to continue working on projects with servo motors and Arduinos myself.

The cooperation between the team members was close to flawless. We had no issues communicating with each other. We worked together in someone's garage, did some discussing at school, or worked

on projects online. We delegated tasks based on everyone's expertise and that worked out pretty well (although this entire project is all Arne's expertise). As for the communication with the tutor, it was not always smooth, but it did not pose any major problems. The tutor did give us useful feedback regarding the report, but in terms of the robot itself, he did not give feedback at all, which we did not really expect, so it was not a problem (our PWS was quite loosely related to physics after all).

One big tip I would give to people doing their PWS next years is to plan ahead for a long period. We did not plan ahead, we eventually came into a bit of trouble because of that. Making a general planning for the entire PWS or just the research phase is very helpful, close to being essential.

Milan

The PWS was a very educational project and I gained a lot of experience during it. While I had done a small project using the Arduino earlier, that was not nearly enough to aid in this PWS. I learned a lot about building robots, as I had no prior experience in that field. Especially the debugging of the robot made me realise how difficult even the simplest things can be. I definitely was the least experienced of the group what made it difficult to contribute due to my very limited knowledge of coding and lack of experience with Arduino. I wish that I had a bit more experience in coding prior to starting the PWS, so I could have contributed to catmesh_pws too. Even though this was the case, I still really enjoyed working on it and, as mentioned earlier, learned a lot. Working with the other team members was especially great, as there were no issues in communicating the plans and planning.

Because the process of building and 'debugging' the robot was so unpredictable there was not a lot of theoretical research to be done, but it did make me realise that replicating a cat's movement was not going to be possible, due to the fact that their walking pattern exceeded the boundaries of what we were capable of with our robot. Combined with the fact that the legs were unlikely to hold the main body it was clear that we were not going to replicate it exactly, but the fact that it still doesn't come as close as we had hoped is rather disappointing.

Within our group we knew exactly what we were going to do and what we were busy with, and I felt that our tutor did not. This is understandable, because this is a very extraordinary project and requires much knowledge into other subjects, many of which also unrelated to physics. Even though for the most part we did not need our tutor's help, when we did, we received response rather quickly. We, however, did not respond that quickly to the two questions he asked.

For any students starting their PWS I would advise them to not underestimate their research and take into account that they can encounter problems which might even cause your research to not have a favourable outcome.

Sources

Adafruit. (2020, January 18). *Adafruit-PWM-Servo-Driver-Library/examples/servo/servo.ino.* Retrieved November 20, 2020, from GitHub: https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library/blob/master/examples/servo/servo.ino

Alexander, D. (2018, November 28). *Biomimicry: 9 Ways Engineers Have Been 'Inspired' by Nature*. Retrieved from Interesting Engineering: https://interestingengineering.com/biomimicry-9-ways-engineers-have-been-inspired-by-nature

Alexander, D. (2018, November 18). *Biomimicry: 9 Ways Engineers Have Been 'Inspired' by Nature*. Retrieved November 20, 2020, from Interesting Engineering:

https://interestingengineering.com/biomimicry-9-ways-engineers-have-been-inspired-by-nature

Arduino. (n.d.). *What is Arduino?* Retrieved November 20, 2020, from arduino.cc: https://www.arduino.cc/en/Guide/Introduction

Barr, M. (2001, September 1). *Introduction to Pulse Width Modulation (PWM)*. Retrieved November 20, 2020, from Barr Group: https://barrgroup.com/Embedded-Systems/How-To/PWM-Pulse-Width-Modulation

Britannica. (n.d.). *Cat: General Features And Special Adaptations*. Retrieved from Encyclopedia Brittanica: https://www.britannica.com/animal/cat/General-features-and-special-adaptations

Choset, M. T. (n.d.). *Bioinspired robots: Examples and the state of the art*. Retrieved from ScienceMag: https://www.sciencemag.org/bioinspired-robots-examples-and-state-art

cplusplus.com. (n.d.). *Compilers*. Retrieved November 20, 2020, from cplusplus.com: http://www.cplusplus.com/doc/tutorial/introduction/

Degeler, A. (2015, December 29). *Marines' LS3 robotic mule is too loud for real-world combat*. Retrieved from ArsTechnica: https://arstechnica.com/information-technology/2015/12/us-militarys-ls3-robotic-mule-deemed-too-loud-for-real-world-combat/

Degeler, A. (2015, December 28). *Marines' LS3 robotic mule is too loud for real-world combat*. Retrieved November 20, 2020, from ARS Technica: https://arstechnica.com/information-technology/2015/12/us-militarys-ls3-robotic-mule-deemed-too-loud-for-real-world-combat/

Elprocus. (n.d.). *Stepper Motor – Types, Advantages & Applications*. Retrieved November 20, 2020, from Elprocus: https://www.elprocus.com/stepper-motor-types-advantages-applications/

Gillis, R. (2002, July 22). *Pectoral girdle and forelimbs*. Retrieved from Zoo Lab: https://web.archive.org/web/20070306082100/http://bioweb.uwlax.edu/zoolab/Table_of_Contents/Lab-9b/Cat_Skeleton_1/Cat_Skeleton_1b/cat_skeleton_1b.htm

Gorgul, E. (2005, September 15). *Shinkansen 500 Kyoto 2005-04-05.jpg.* Retrieved from Wikimedia Commons: https://commons.wikimedia.org/wiki/File:Shinkansen_500_Kyoto_2005-04-05.jpg (This file is licensed under the Creative Commons Attribution 2.5 Generic license: https://creativecommons.org/licenses/by/2.5/deed.en.)

ISO/IEC. (2013, October 13). *Working Draft, Standard for Programming Language C++*. Retrieved November 20, 2020, from open-std.org: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3797.pdf

ITead Studio. (2010, November 3). *Ultrasonic ranging module : HC-SR04*. Retrieved November 20, 2020, from electroschematics.com: https://www.electroschematics.com/wp-content/uploads/2013/07/HC-SR04-datasheet-version-2.pdf

Julian F.V Vincent, O. A.-K. (2006, April 18). *Biomimetics: its practice and theory*. Retrieved from The Royal Society Publishing: https://royalsocietypublishing.org/doi/10.1098/rsif.2006.0127

Keats, D. (2013, April 27). *Malachite Kingfisher, Alcedo cristata at Marievale Nature Reserve, Gauteng, South Africa.* Retrieved from Flickr: https://www.flickr.com/photos/93242958@N00/8688965285 (This file is licensed under the

Creative Commons Attribution 2.0 Generic license: https://creativecommons.org/licenses/by/2.0/deed.en.)

NXP Semiconductors. (2014, April 4). *UM10204 I2C-bus specification and user manual*. Retrieved November 20, 2020, from nxp.com: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

NXP Semiconductors. (2015, April 16). *PCA9685 16-channel, 12-bit PWM Fm+ I2C-bus LED controller*. Retrieved November 20, 2020, from nxp.com: https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf

Protalinski, E. (2020, June 16). *Boston Dynamics starts selling its Spot robot — for $74,500*. Retrieved from VentureBeat: https://venturebeat.com/2020/06/16/boston-dynamics-buy-spot-robot-74500/#:~:text=Spot%20robots%20have%20been%20used,construction%20sites%2C%20and%20research%20laboratories.

Protalinski, E. (2020, June 16). *Boston Dynamics starts selling its Spot robot — for $74,500*. Retrieved November 20, 2020, from Venturebeat: https://venturebeat.com/2020/06/16/boston-dynamics-buy-spot-robot-74500/#:~:text=Spot%20robots%20have%20been%20used,construction%20sites%2C%20and%20research%20laboratories.

SERVO MOTOR SG90 DATA SHEET. (n.d.). Retrieved November 20, 2020, from ee.ic.ac.uk: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

Sharma, B. (2020, October 16). *What is LiDAR technology and how does it work?* Retrieved November 20, 2020, from Geospatial World: https://www.geospatialworld.net/blogs/what-is-lidar-technology-and-how-does-it-work/

Travers, M., & Choset, H. (n.d.). *Bioinspired robots: Examples and the state of the art*. Retrieved November 20, 2020, from Science: https://www.sciencemag.org/bioinspired-robots-examples-and-state-art

Vincent, J. F., Bogatyreva, O. A., Bogatyrev, N. R., Bowyer, A., & Pahl, A.-K. (2006, April 18). *Biomimetics: its practice and theory*. Retrieved November 20, 2020, from The Royal Society Publishing: https://royalsocietypublishing.org/doi/10.1098/rsif.2006.0127

Web Summit. (2019, November 7). *HM2_9609.* Retrieved from Flickr: https://www.flickr.com/photos/74711243@N06/49028927416 (This file is licensed under the Creative Commons Attribution 2.0 Generic license: https://creativecommons.org/licenses/by/2.0/deed.en.)

Wilson, J. (n.d.). *Bones of the Cat-All About The Cat's Skeleton*. Retrieved from Cat-World: https://cat-world.com/bones-of-the-cat/

ZooLab. (2002, July 22). *Pectoral girdle and forelimbs*. Retrieved November 20, 2020, from ZooLab (via the web archive): https://web.archive.org/web/20070306082100/http:/bioweb.uwlax.edu/zoolab/Table_of_Contents/Lab-9b/Cat_Skeleton_1/Cat_Skeleton_1b/cat_skeleton_1b.htm