Semi-Autonomous robotic arm

EPQ 2024-2025

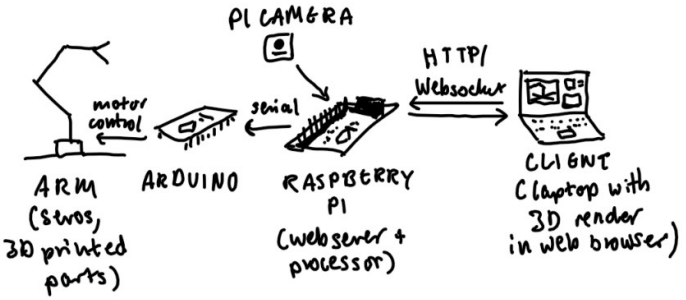
Contents

1 General

1.1 Overview

1.1.1 Summary

The overall aim for this project is to design a high precision robotic arm and provide easy to use controls for it, including some autonomous capabilities. An important part of this is the digital twin, which I have implemented as a 3D render of the arm's current position and has various positioning controls. The project also has a substantial computer vision aspect, with object and colour detection to produce size and position calculations. The objects are then rendered in their position relative to the arm, with the option to pick them up. The final major software aspect is writing scripts, meaning that the user can write programs in a simple format to control the arm autonomously.



Different components of the design

The arm itself is designed in Fusion 360 and 3D printed, controlled by an Arduino and powered with various servos and steppers. This is connected to a raspberry Pi, which hosts a Python webserver that sends commands to the Arduino over a Serial port and creates a WebSocket connection to the HTML that it also serves. The rendering is done using WebGL and TypeScript, and the documentation for writing scripts is also served by the python webserver, written with markdown and formatted using MkDocs.

1.1.2 Initial project objectives

Before starting this project, I made a list of what I wanted to achieve. This influenced my overarching design decisions and project management, as it broke the project as a whole down into more manageable chunks. I expected my result to be different from my initial objectives as I was not initially sure how difficult some tasks would be, although I did end up achieving most of them.

1.1.2.1 Hardware / lower-level code

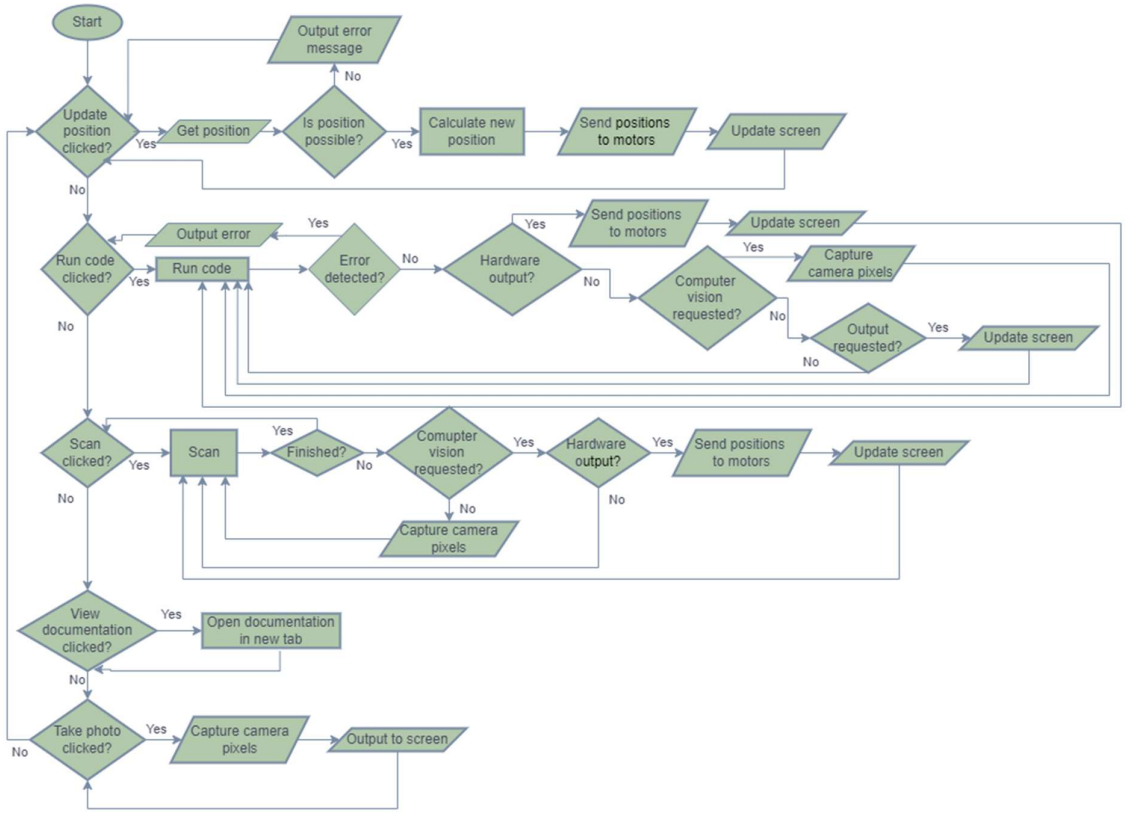
1. Physical arm
    a. To create a reliable arm that can operate for extended time periods.
    b. To have the ability to send a microcontroller motor movements and the motors rotate to the expected position.
    c. To provide functionality for objects to be picked up.
    d. To allow the arm to rotate 360 degrees at the base, having at least 180 degrees of freedom on each joint.
2. Positioning
    a. Arm reaches expected position with the claw parallel to the ground.
    b. Arm is aware of where all parts of the arm are.
    c. Arm does not attempt to move beyond the servo's degrees of movement.

1.1.2.2 Client

3. 3D rendering
    a. To create an interactive display of the arm's position and claw state.
    b. To allow the user to rotate and change position through an overlayed UI
    c. To render detected objects on the display
    d. To allow user to click on detected objects to pick them up
    e. To display a notification if desired action is impossible
4. Image forwarding
    a. To get images from the Pi when requested
    b. To display images in another part of the client's screen
5. Design paths
    a. To allow the user to write basic scripts
    b. To provide debugging functionality for the user
    c. To allow the user to run these scripts

1.1.2.3 Server

6. Client communication
    a. Server receives image requests from client
    b. Server receives position commands and sends new beam locations
    c. Server sends information about detected objects to the client
7. Object detection
    a. To detect objects of a known size (single camera) and find their location with a tolerance of ±10mm
    b. To detect object that have been moved, and retry any failed pickup
8. User defined scripts
    a. To robustly parse user defined scripts
    b. To provide user with mechanisms to debug their scripts
    c. Server has ability to run the scripts directly on the arm
9. Web
    a. To serve both WebSockets and HTML
    b. To move the arm based on commands
10. Images
    a. To use a Picamera that can take images on client's request
    b. To forward images to client or store them for computer vision



Flowchart

1.1.3 Overarching design choices

I left most of the specific technical choices (e.g. type of stepper, choice of HTTP library) to the point in the project where I had enough background to pick the best option for my precise requirements. The research behind these is detailed in the relevant design sections of the document. My initial choices were as follows:

1.1.3.1 Computer aided design (CAD) software

As computer aided design was a major section of the project, I decided to research different CAD software to pick which one is most appropriate for my project. The options I picked were Onshape, Fusion 360 and SolidWorks as these are all very common and have different benefits.

	Onshape	**Fusion 360**	**SolidWorks**
Parametric functionality	Medium	High [1]	Very high
Prior knowledge	No	Yes	No
Ease of use	Easy	Medium	Hard
Student license	Yes	Yes	No

I decided on Fusion 360 as it has more features than Onshape, and prior experience with it meant ease of use was less of an issue. While SolidWorks would have been a better choice due to the increased functionality, the lack of student license made this infeasible.

1.1.3.2 Types of motors

The main overarching groups of motors are Servos, stepper motors and DC motors (brushed or brushless). As these categories are very broad, most of my research on them is qualitative.

	Servo	**Stepper**	**DC**
Degrees of freedom	180°	360°	360°
Control mechanism	PWM signal [2]	Dedicated driver	MOSFET and on/off pin
Precision	Absolute	Relative to 2°	Low
Torque	Medium	Low	High
Cost	Medium	High	Low-medium
Weight	Medium	High	Low
Speed	Medium	Low	High

Due to the accuracy required and that I did not have requirements to lift anything heavy, I immediately discounted DC motors. I did not need any joint in the beam to turn more than 180 degrees, and I wanted the arm to change position relatively quickly, so I decided to use servos for these joints. For the base, I wanted something that turned a full 360 degrees accurately, so I decided to use a stepper motor for this. The disadvantage of the stepper being heavy also became an advantage in this situation, as it provides extra stability.

1.1.3.3 Microcontroller

The requirements for the microcontroller were fast, reliable communication with the raspberry Pi and the ability to control 4 servos and a stepper, meaning that it needed at least 4 PWM pins and 2 other normal pins for the stepper. I decided on an Arduino nano [3] for this purpose due to its compact size, serial communication on the USB port and 6 PWM pins. These also have a lot of online resources, making debugging easier, and I had used them enough previously to know that they would work for this purpose.

1.1.3.4 Server hardware

For the server, I wanted to use something reasonably small and inexpensive, as I did not want it to consume too much power. It also needed to be powerful enough to host the webserver, although this is not very computationally expensive as the 3D rendering is handled by the browser. Due to how common tutorials and documentation are, I narrowed this down to a raspberry Pi [4] and decided against a Pi Zero due to lack of processing power. I decided on a Pi 4 as this seemed like a reasonable balance between power consumption (why I decided against the Pi 5) and speed.

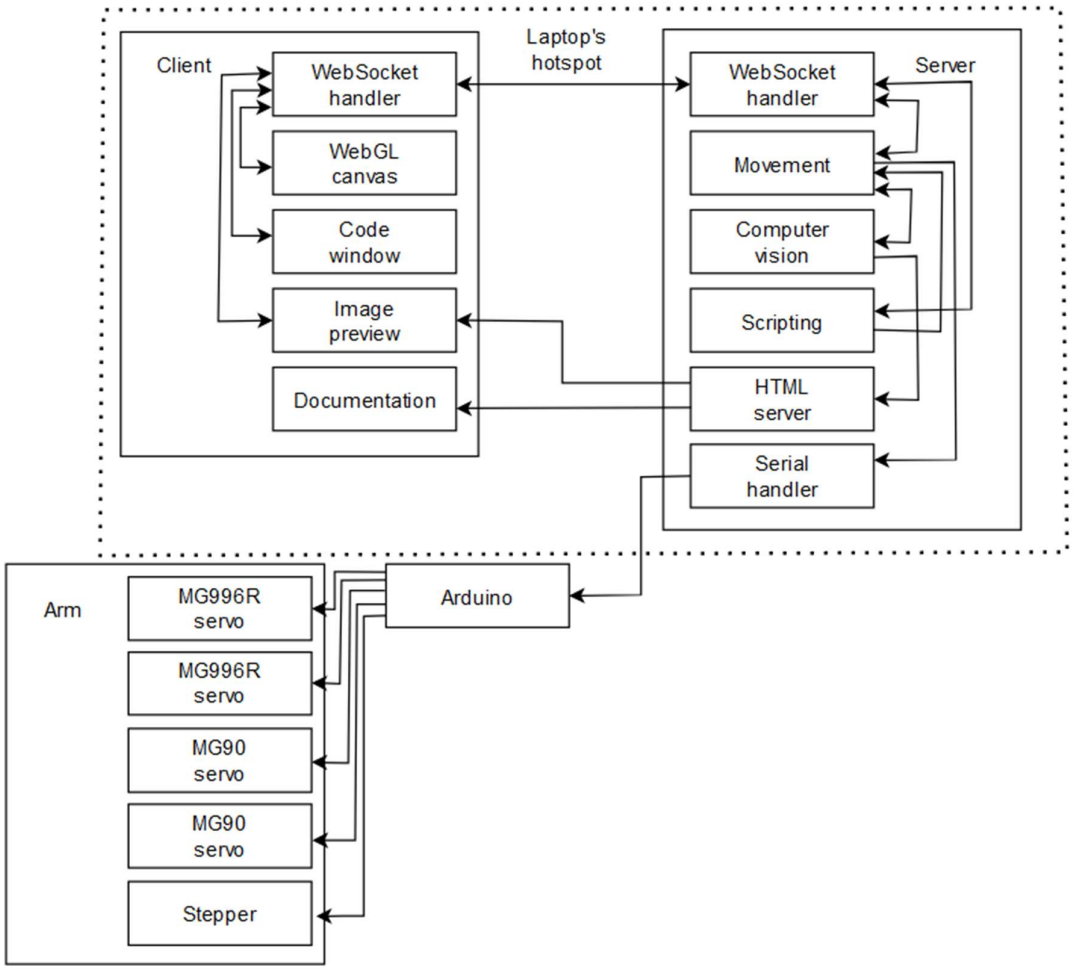
1.1.3.5 Server language

Python is generally the default language to use when programming raspberry Pis as it built into the operating system, so I did not fully consider many other options as I saw no drawbacks to using it. Prior to starting this project, I had written a Python webserver once before, but at the time I did not consider the libraries that I wanted to use and how they linked together, causing quite a few issues later in the project. This meant that I was interested in how I could do a Python webserver better, using different libraries, which was another motivation for picking Python. Python is also my strongest programming language, and I thought that using something I am comfortable with for the main part of the programming would leave me with more time for more interesting technical questions.

1.1.3.6 Client language / frameworks

The only real choice for writing the client scripts was JavaScript, unless I wanted to investigate web assembly or similar. However, in previous JS projects, I found JavaScript's weak typing system [5] (all variables have a flexible type that is implicitly converted when required) hard to debug, so I wrote the code in TypeScript. This is JS with additional typing syntax and code editor support, which is then converted (transpiled) into JS code [6]. As I had found this useful in the past, I started writing in TS from the beginning.

I also decided on the rendering API before starting writing the client code, as influences the majority of what I planned to write. The most common options for this are WebGL and Three.js. WebGL is a relatively low-level graphics interface, based of OpenGL, and Three.js is a higher-level library built on top of WebGL [7], making it simpler to use but also significantly abstracted from the actual rendering process. The website states that these are all positives, but given it is clearly advocating for the use of three.js I did not consider these opinions valid. As my plan for the rendering was relatively simple, I did not think complexity would be too much of an issue, and I hoped to be able to learn more about rendering through the course of this project. I had also used WebGL before for a very different project, so a lot of the initialisation and buffer code was the same.

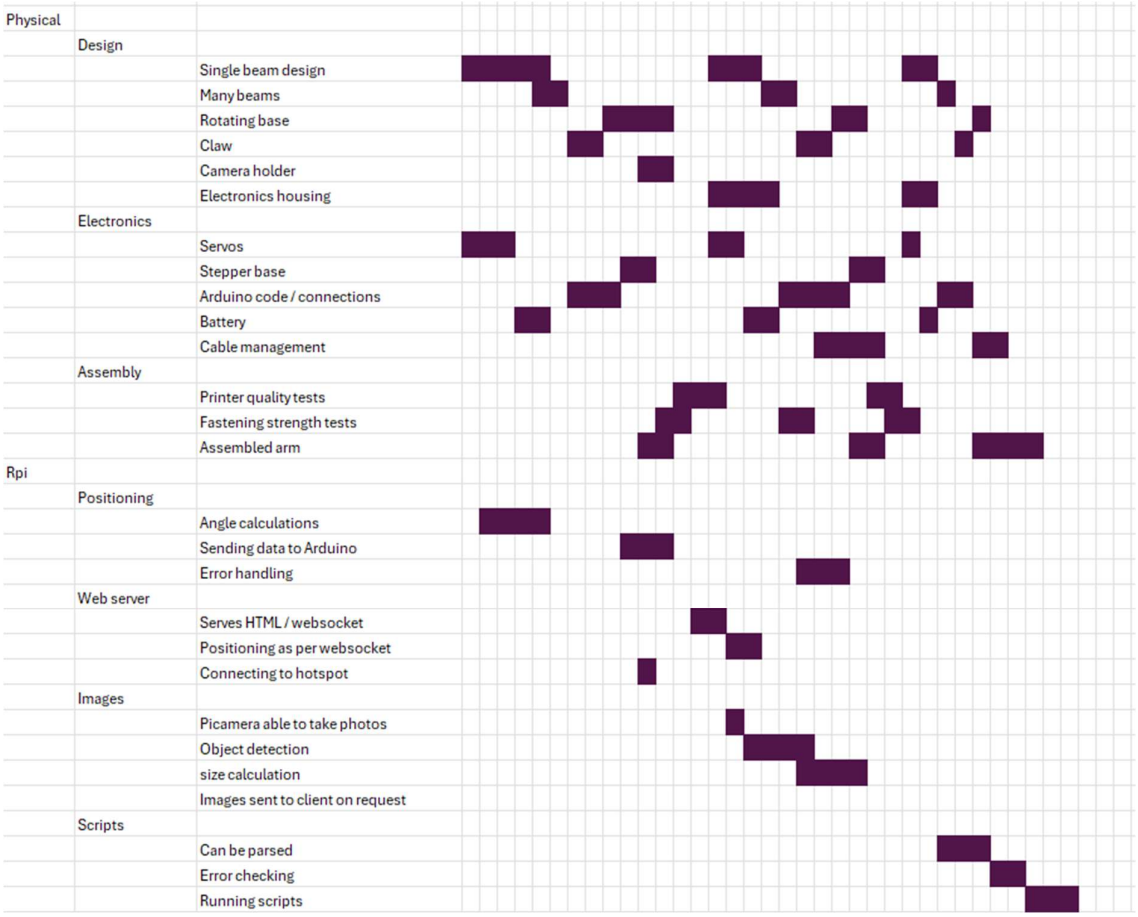


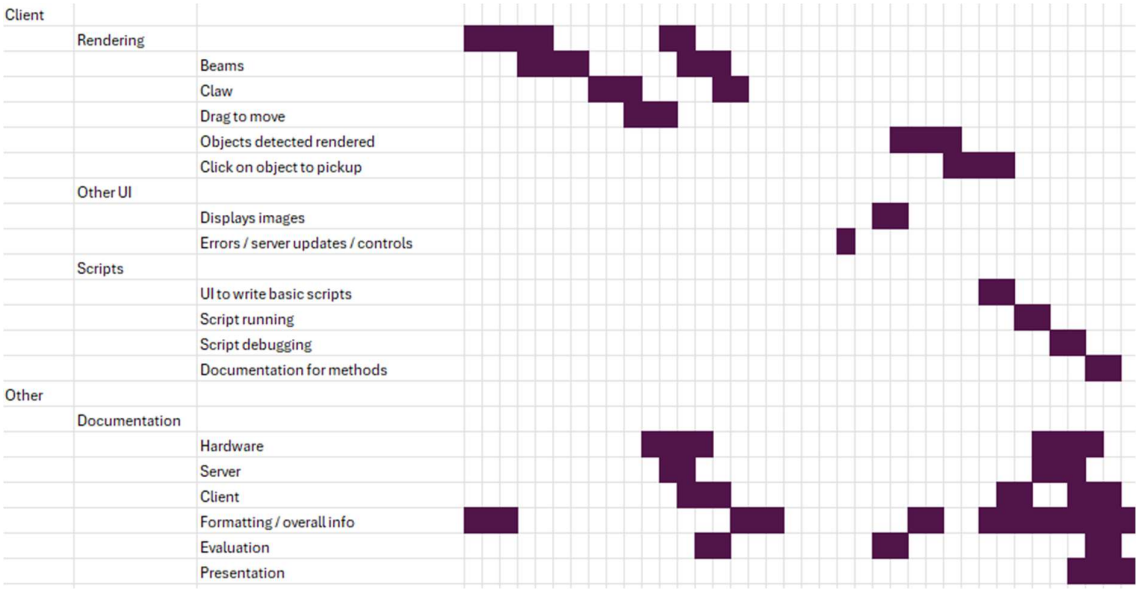
High-level technical diagram

1.2 Planning

1.2.1 Gantt chart

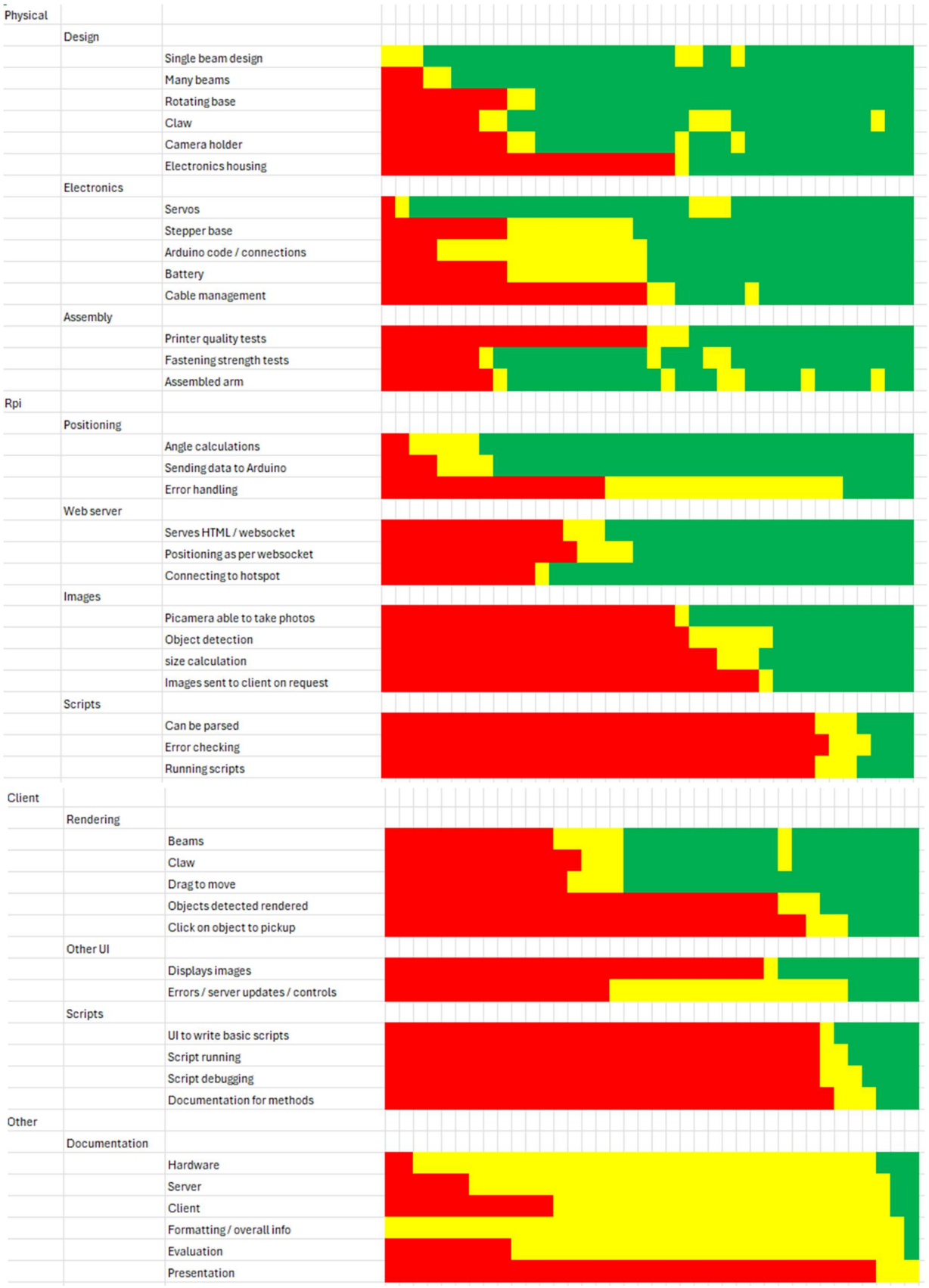
I created a Gantt chart with all the required tasks, showing when I would do each one. Many tasks have multiple sections, as I expected this project to be an iterative process. I chose to not update this chart dynamically, instead referring to it as I updated the milestone tracker to ensure any part was not being left behind. This was so that in the process of updating the charts I did not lose sight of my initial plan for timescales and would continue to allocate enough time for some of the later tasks.





1.2.2 Milestone tracker

I used this to track my progress throughout the project, as I expected when I finished tasks to deviate from my initial plan. I marked sections as completed once I thought they were finished, but I often realised that they needed more improvement, resulting in them moving back to “in progress” later.


Physical
Design
Single beam design
Many beams
Rotating base
Claw
Camera holder
Electronics housing
Electronics
Servos
Stepper base
Arduino code / connections
Battery
Cable management
Assembly
Printer quality tests
Fastening strength tests
Assembled arm
Rpi
Positioning
Angle calculations
Sending data to Arduino
Error handling
Web server
Serves HTML / websocket
Positioning as per websocket
Connecting to hotspot
Images
Picamera able to take photos
Object detection
size calculation
Images sent to client on request
Scripts
Can be parsed
Error checking
Running scripts
Client
Rendering
Beams
Claw
Drag to move
Objects detected rendered
Click on object to pickup
Other UI
Displays images
Errors / server updates / controls
Scripts
UI to write basic scripts
Script running
Script debugging
Documentation for methods
Other
Documentation
Hardware
Server
Client
Formatting / overall info
Evaluation
Presentation

1.2.3 Project log

Date	Comments
20/09/2024	Started writing project proposal, decided on the languages / software I was going to use
27/09/2024	Finished project proposal, considered requirements for the beam and researched and chose servos.
04/10/2024	Looked at different joint designs and designed and printed one.
11/10/2024	Discovered some issues with previous joint and reprinted a working version. I also looked at articles relating to how to do angle calculations
18/10/2024	Designed a method of connecting the joints together and printed out a couple, derived the inverse kinematics from the articles.
25/10/2024	Tested the joint connection and printed a third beam, plus a static base. I also wrote code to control the servos via the Serial monitor on the Arduino IDE
01/11/2024	Wrote python code to perform the inverse kinematics, and added a simple Tkinter GUI
08/11/2024	Improved the code I wrote last week and added a Serial link to the Arduino
15/11/2024	Tested the code and realised that I needed to use PETG for the gears, so reprinted. I also started designing the claw.
22/11/2024	Basic claw design finished / printed and reassembled with PETG gears. I also added open/close claw functionality to the GUI.
29/11/2024	Started researching steppers and designing the base. In the process of this, I also investigated power management and options for batteries
06/12/2024	Continued researching steppers and power management, finished designing the base
13/12/2024	Ordered the stepper and set up a Raspberry Pi connecting to a hotspot
20/12/2024	Started the WebGL render
27/12/2024	Continued working on the rendering of a beam, attempted the drag to move
03/01/2025	Finished the drag to move, created a WebSocket channel to the server, rendered the claw
10/01/2025	Added functionality for the server to serve HTML, added position textbox than on update sent a serial command
17/01/2025	General debugging of the above code and moving it to a Pi
24/01/2025	Looked into how to control the stepper, tried a few different drivers and power management
31/01/2025	Successfully controlled the stepper, printed and assembled the base
07/02/2025	Attempted to reprint gears with Nylon due to wear, so tested different print settings. Printed the gear and realised Nylon is too flexible, made a circuit diagram for the electronics
14/02/2025	Repeated the above process with ASA (significantly better), started re-assembling the entire arm with this, soldered the electronics
21/02/2025	Further testing and calibration, attached a Picamera and got it taking photos, designed an electronics case

28/02/2025	Discovered that the lower joints were underpowered so researched different servo options, attempted basic contour detection
07/03/2025	Decided on a servo and redesigned / reprinted the base joint for this, redesigned the claw with TPU inserts
14/03/2025	Reassembled the arm with new (after a few iterations) claw and base joint, researched distance calculation and took sample images
21/03/2025	Got the sample images working mostly with distance calculations (although with an unexpected formula), reprinted the Picamera mount
28/03/2025	Printed sample barrels, redid distance calculations with real camera and barrels, added multicolour contour detection
04/04/2025	Joints continued slipping, so I did an FEA analysis to find the weak points and investigated different ways of strengthening this part, wrote the view image part of the client and modified render to work with different colours
11/04/2025	Redid the joints with stronger connector, improved render of all the components and attempted to write scan code
18/04/2025	Finished scan code and tested / calibrated it, realised that another servo was too weak so redesigned this joint and reprinted the entire arm slightly larger, and a few modifications to electronics case
25/04/2025	Assembled entire arm / resoldered electronics neater, investigated click to pick up code
02/05/2025	Wrote + debugged click to pick up code, investigate parsers and wrote client-side code
09/05/2025	Calibration of pickup and scan code, some backend parser code, looked into documentation, server updates / error checking
16/05/2025	More calibration and parser code, started writing documentation for the parser
23/05/2025	Finished writing documentation for the parser, even more calibration / debugging, finished parser code
30/05/2025	Finished incomplete sections of this document and proofreading, started writing the presentation and recording the arm while it works, final redesign / calibration of claw
06/06/2025	Writing the presentation, final proofreading pass of overall document
13/06/2025	Finalising / rehearsing the presentation

2 Physical arm

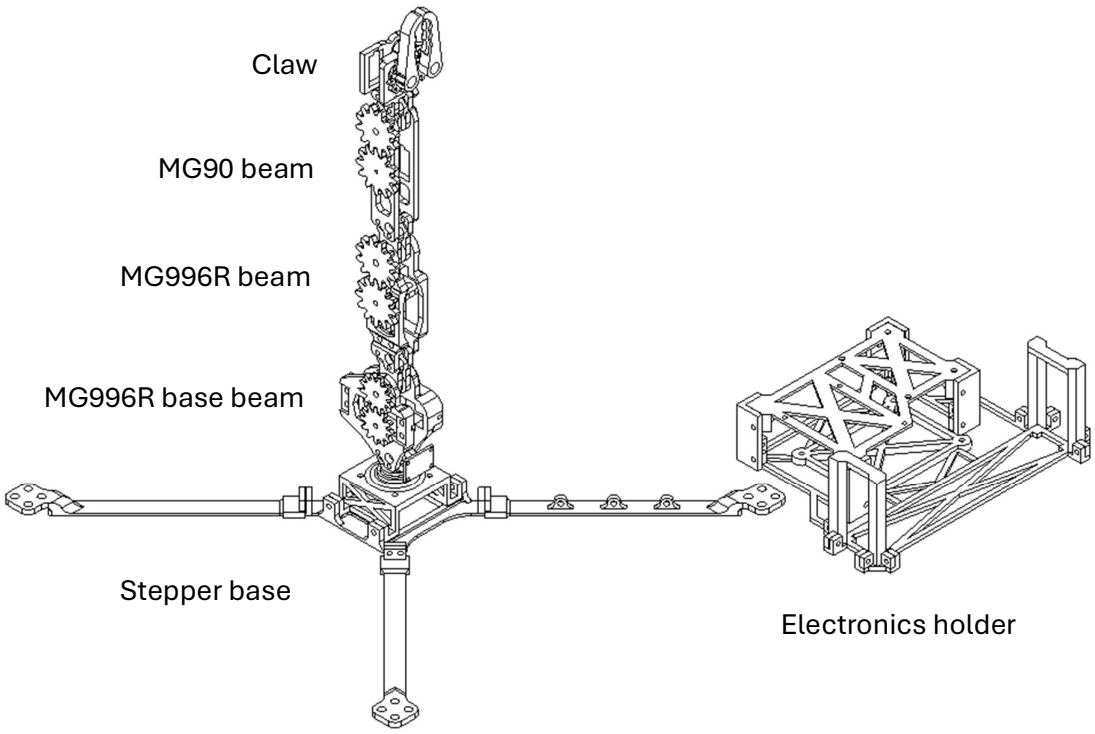
2.1 Design

2.1.1 Overall design choices

At a minimum, the arm needs to have a claw that can move to a position and open / close there. To move the claw to a position, I decided to connect a series of beams together, each of which added a degree of freedom. Most of these were two dimensional for simplicity, but I put the base on a stepper to add the third dimension.

For simplicity, I decided to duplicate a single connection (beam), three times, each of which have a rotatable joint. The final connection would go to a claw rather than a beam, but the connector would be the same at every joint so that I could simply duplicate the part three times for each beam. As outlined below, I realised later that this would not be possible, but approaching the design from this direction still increased prototyping speed as it made parts easier to change.

I also needed to design the base, which needed to be heavy enough to prevent the arm from tipping over, and electronics holder to securely house the wiring.



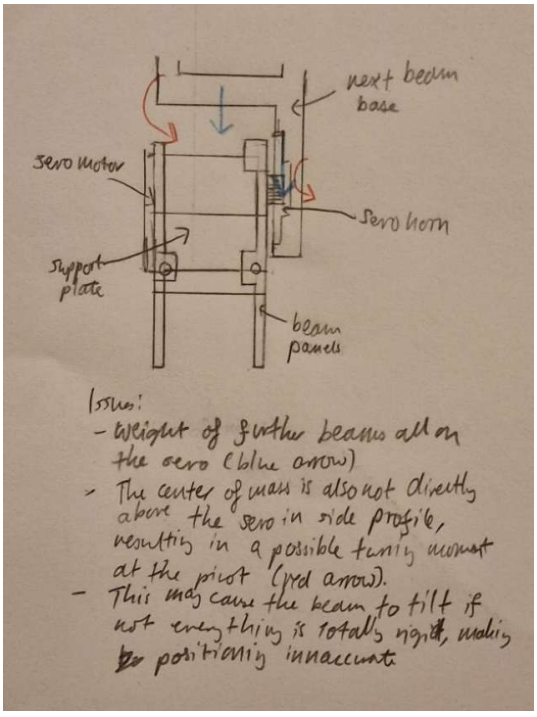
Final arm design for reference throughout this section

2.1.2 Beam

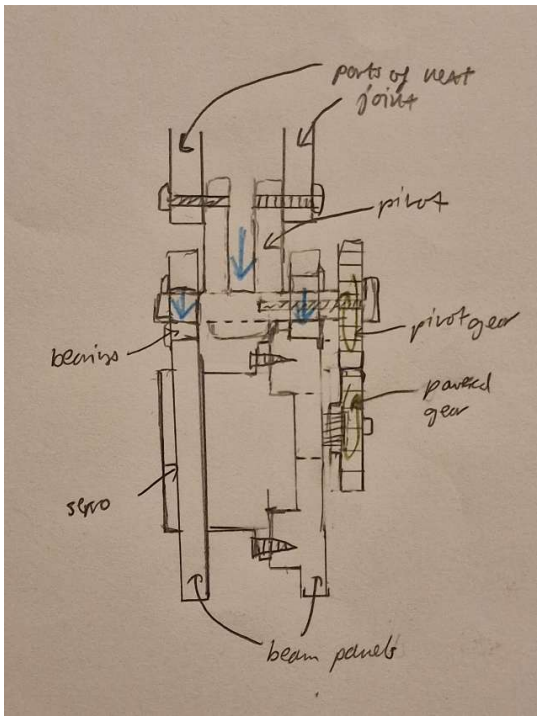
In this design, I wanted to make repairing parts as easy as possible and minimise testing required. To do this, I made the beams modular, meaning I could reuse parts. It also meant that I had multiple identical printed parts, making repairs easier. The focus of this design was reliability, as the arm needed to mirror the digital twin accurately.

One of the causes of inaccuracy is the servo slipping, which is particularly an issue with the MG90 due to the metal gear wearing down most 3D printed plastics, which have comparatively low abrasion coefficients, causing them to wear down quickly. My solution for this was to use the servo horn and a screw as an attachment, as it has a large contact area from the horn to prevent wear to the 3D printed part.

Another cause of inaccuracy is the force caused by a heavy item directly pivoting on the servo (illustrated in first sketch below). This distributes the weight of the beams unevenly, resulting in the servo pivoting on the wrong axis and tilting. I avoided this issue by using gears, with the servo turning a gear, causing another gear with the actual joint to rotate. This other gear can then be supported by bearings, distributing the mass evenly. I decided to use captive flanged bearings as I had access to them and they are easy to press fit.



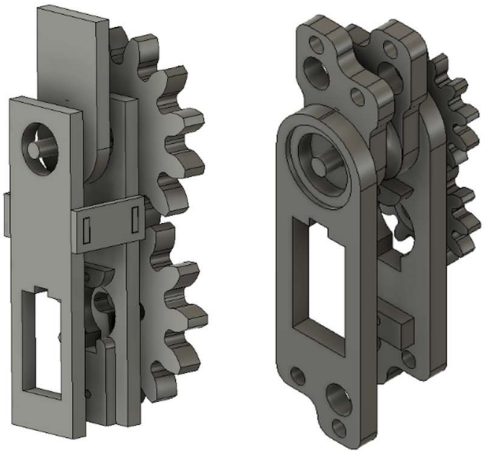
Sketch of initial idea, illustrating problems with weight distribution



Sketch of design (not to scale and ignoring manufacturing constraints) showing how the weight distribution issue could be fixed

This initial design was to test if this joint mechanism works, ignoring the actual connections. The results of this were mostly successful, with no significant issues.

To turn this joint into a beam, the next part to test was the connections between joints. I designed this as shown to prevent accidental rotation, as there are 2 screws per side, arranged so that both sides have similar force. At this point I also realised that the servo horn was not required, as the screw was what actually caused the gear in place. This allowed me to shrink the design, as the distance between the centers of the gears could be smaller. After fixing all of the above issues, I then printed the new design, which worked well.



Initial beam design (without connectors)

Modified design after some testing and with connectors

I later decided to switch to the MG996R servo due to lack of torque (see electronics section). These are larger and heavier, so I only wanted to redesign the first joint as that did not affect the force on any other servo and did not reduce the freedom of movement. Due to the larger size, this also required increasing the distance between pivot and the center of the servo, resulting in larger gears. To maintain freedom of movement, I increased the gear size for all joints.

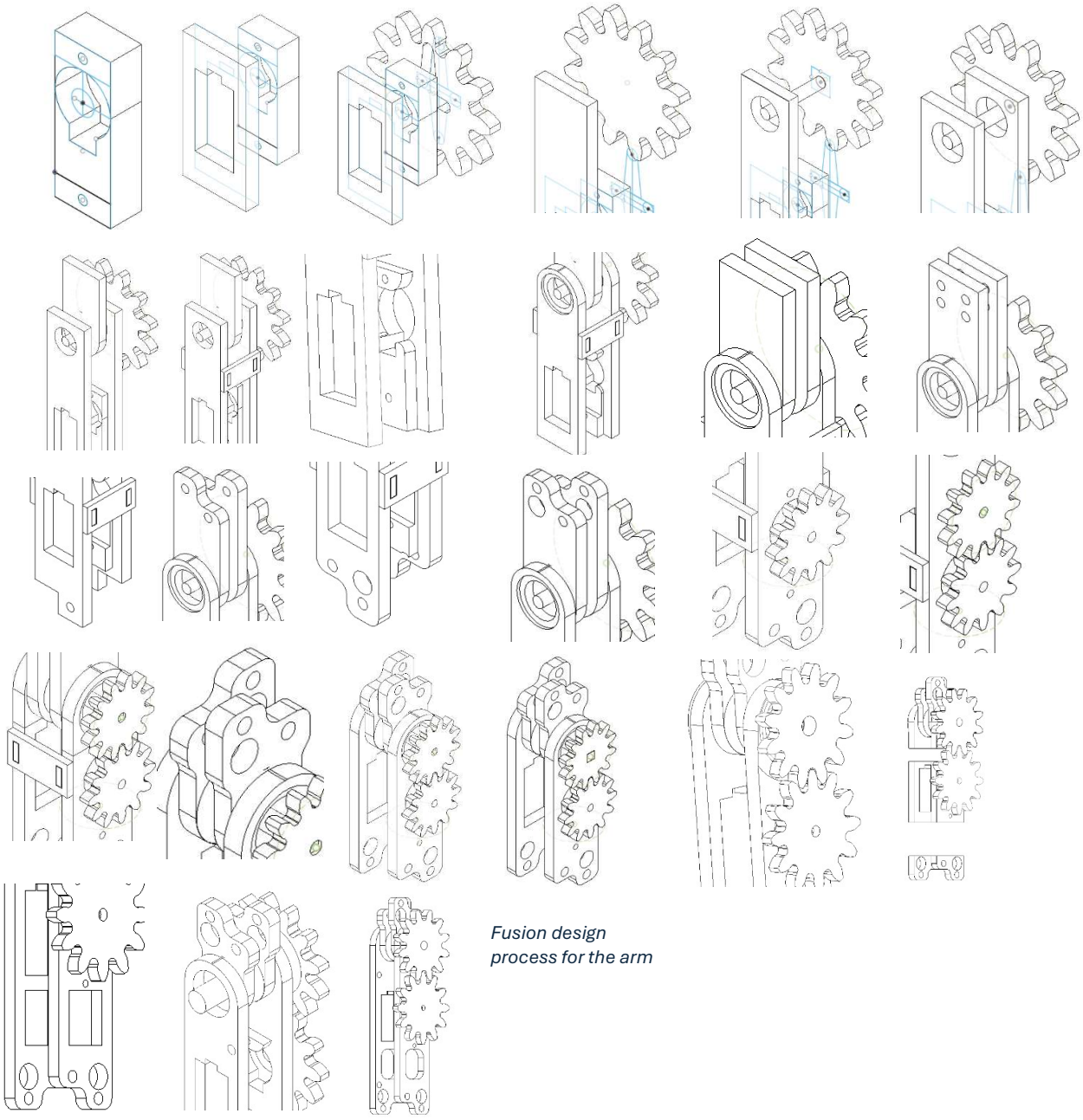
Due to the MG996R's significantly larger size it is mounted sideways, so the beam does not need to be any longer below the gears. There is also more, thicker plastic and bolts are used rather than self-tapping screws to secure it in place as it needs to support more weight that is not at the centre of mass. The connector is still the same, meaning the design is still modular.

Beam modified for the MG996R servo

After more testing I realised that the axle was not strong enough, so I changed the square axle that is part of the pivot to a circular hole going through for a metal axle to be glued into during assembly (see material section).

After some further testing, I decided to switch the second joint to an MG996R as well as this was struggling to hold its position and was frequently overheating, while the first MG996R seemed to have no issue with the current weight. I could not mount the MG996R sideways as this would prevent the joint from rotating backwards, so I needed to significantly increase the beam's length, increasing the later ones by the same amount. I kept the first one the same length as I wanted the first joint to be as close to the ground as possible to increase the range of positions of the arm at ground level.

MG996R beam with motor mounted upright

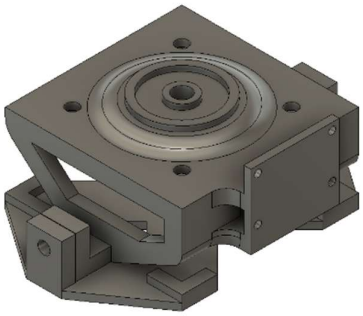


Fusion design process for the arm

2.1.3 Rotating base

The main requirements of the base are to hold the stepper, to not tip over and provide a smooth surface for the rest of the arm to rotate on.

Initial pivot



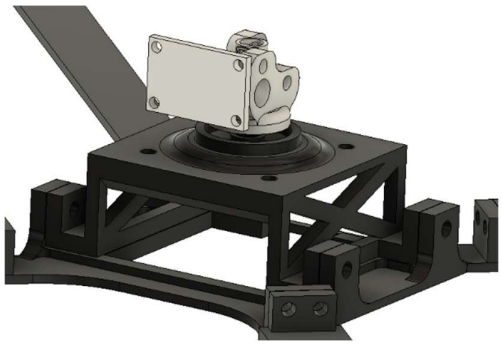
Initial rotating base

The first iteration of the design is shown. For the connection with the arm, I used a thrust bearing, as they allow free rotation between the components while supporting weight [8]. The pivot is also designed to fit on the D shaft of the stepper, although this was slightly too long, and I initially did not realise that the camera needed to rotate as well.

Arms for ballast

After testing, I found that the base was not heavy enough to prevent the arm from tipping over, so I designed 4 "arms" to extend out, spreading out the mass. Each of these has 4 M5x16 screws on the end, acting as ballast. Reprinting the stepper base with these modifications worked well.

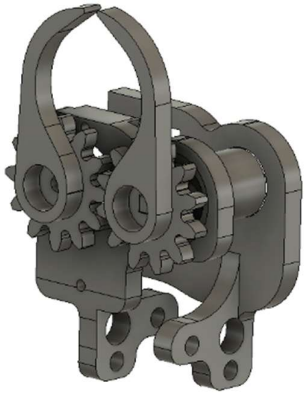
After testing, I noticed a new issue was that the stepper cover could rotate with the base itself due to only having 2 screws, especially noticeable when the stepper heated up and softened the plastic. I therefore redesigned the connection with 4 screws, but still leaving as much air gap as possible to keep the stepper cool. Reprinting this then resulted in a stable stepper base.



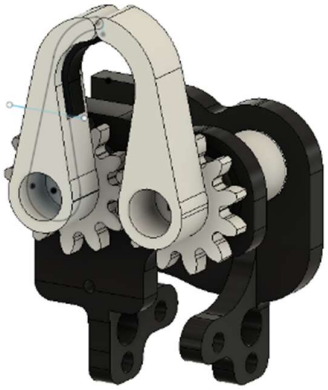
Stepper base

2.1.4 Claw

The aim of the claw is to firmly grip a cylindrical object, 15mm in diameter. I designed this by basing it off another beam but flipping the servo and cogs section 90 degrees and placing the gears in parallel.

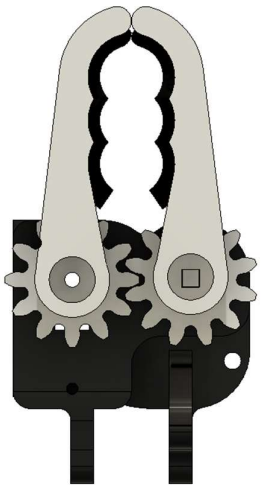


Initial claw design



More secure claw with TPU insert

While this proved all the connections worked, it did not successfully grip the barrel, so I made the claw part thicker for more contact area. I also hollowed out the centre of each claw and added a TPU insert with no solid perimeter and high infill (see print quality) that could be compressed slightly, increasing the grippiness.

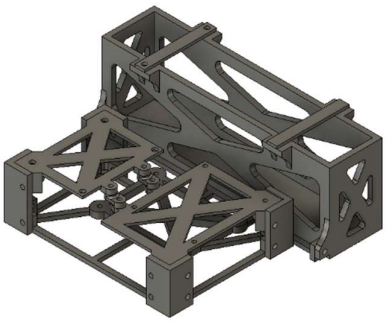


Larger claw

One issue I discovered with this design was that it would only successfully grab the barrel if it was in the correct place, with <5mm room for error on computer vision and positioning. This positioning accuracy would not be possible in the scope of this project, so I decided to lengthen the claw, meaning that my distance and position calculations could be slightly less accurate as it would work for barrels it was not entirely centred on and had a distance tolerance of +-7.5mm

2.1.5 Electronics housing

There were two possible styles I looked into for creating this holder: one which was a solid box with all the components inside or a minimalist frame that just provided structural support. I chose the second option, as while the first one has the advantage that components are less likely to be accidentally disconnected and the outcome may look neater, it uses far more plastic and makes debugging the circuit harder.



Initial electronics housing design

This initial design works and was structurally sound, although I never printed the battery holder due to the amount of filament required and the number of supports that would defeat the point of the gaps. I therefore redesigned it into part of the base, with 4 beams around the edge and 2 covers for the corners without connectors. This had a far lower print time and successfully secured the battery.

New design for the battery holder

Stronger design

While this worked, I found that after continuously transporting the case around, the screws between the two upper layers came undone. As there was no requirement for the voltage converter and raspberry pi to be at different levels, I raised the lower one and printed them both as one part, removing the need for this connection. I also modified the battery holder to be slightly more secure by adding a beam across the top on the short sides. As the plastic can be slightly bent, it is still possible to change the battery just by bending the cover away.

2.2 Electronics

2.2.1 Servos

I chose to use servos for each of the joints due to their high precision and because they rotate relative to a known position rather than their last movement. Initially, my options were limited to the SG90 and MG90S, as these were what I had access to and I wanted to get a proof of concept working before ordering any new components.

	MG90S [9]	SG90 [10]
Weight	13.4g	9g
Torque at 4.8V	1.8kgf/cm	1.3kgf/cm
Dimensions	~22x12x28mm	~22x12x35mm
Operating voltage	4.8-6V	4.8-6V
Gear material	Metal	Plastic
Price	~£2 / unit	~£1.50 / unit

These servos require enough torque to lift the weight of the rest of the arm. Out of these two options I chose MG90S, as it has a 38% increase in torque, but the total mass difference is around 11% due to the constant weight of the plastic / screws (~30g). Another benefit of this choice is that the MG90S is more durable due to the metal gears, although there is a risk that this causes other components to wear down faster. (see 2.1)

After further testing with the complete arm, I then found that while these servos worked well for the further out joints, the first one often did not have enough torque as it had to support the entirety of the arm. The requirements for the base servo are slightly different to the others, as weight is not an issue and less freedom of movement is required, meaning the servo can be slightly larger. It could also be slightly more expensive, although I wanted to avoid any over £10/unit as those seemed too specialised for my purpose. To keep power simple I still wanted a servo that could run off 5V.

The servo that I found for this was the MG996R, which is frequently used in slightly larger robotics projects. With a far higher torque of 9.4 kgf/cm, it weighs 55g and is 40x20x43mm [11], although neither of these are an issue in this case as the only component that needs to rotate it is the stepper. It is also not too expensive, at £4/unit. Testing this servo on the modified design was successful, and it did not require any circuitry changes as it is controlled in the exact same way.

MG996R

2.2.2 Battery

The main considerations for this are voltage, max current draw, cost and safety. Each servo draws 400mA (before switch to MG996R, which has an operating current of 500mA and stall current of 2.5A), and the pi draws up to 600mA [12], all at 5V. This means the maximum current draw from the Pi is 1.2A (excluding the stepper), which is too close to the maximum that can be drawn from the peripherals (exactly 1.2A), which is what I was using for initial testing. Steppers are also often 12V so a useful requirement would be for the battery to be able to provide both 5 and 12V.

After researching the batteries I had access to, I found a camera battery that could provide both 14.8V (quickly drops down to 12) and 5V. This addressed the voltage consideration, but the 5V supply could provide a maximum of 2A only, not enough for the servos. It also had a unnecessarily high capacity of 99Wh [13], making it bulky and expensive.

I then investigated using a LiPo battery and a voltage regulator. This in theory would work as the pi / servo circuit could be powered via the voltage regulator, and LiPos can provide more than enough current. I decided against this as I do not need something particularly light, and the safety considerations with working with LiPos meant it was not worth it. [14]

The battery that I ended up using was a small 12V 25Wh lead acid battery [15] as it was safe, comparatively cheap and had enough capacity and current draw. Compared to other types of battery, lead acid has a lower capacity to mass ratio, although in my use case this is not an important consideration. The only issue was voltage, so I used a buck boost converter [16] (a type of voltage regulator) to power the rest of the circuit, which could reliably power the Pi.

2.2.3 Stepper

I decided to use a stepper to rotate the base as it needs 360 degrees of movement and to be stable enough for the rest of the arm. Homing is less of an issue as calibration can just be done by eye when the arm is turned on.

My initial thought was to use a 28BYJ-48 [17] stepper motor due to how common they are. However, these have a very low torque and while I aimed for there to be very little rotational force on the stepper I decided to go straight to a higher rated one.

I decided to use a standard NEMA stepper due to the ease of mounting. I also went for a pancake stepper (less depth) to not add extra height to the arm as this decreases the horizontal range. NEMA sizes are the size of the base of the stepper in inches x 10 [18], ranging from 8 – 42, with 17, 23 and 34 the most common. I went with NEMA 17 as I did not need anything particularly large but still wanted the range of suppliers that come with using a standard size. I also went for a 12V stepper [19] as it could bypass the voltage regulator part of the circuit. This has a torque of 17Ncm, a major improvement from the 28BYJ-48's 34.3mNcm.

NEMA 17 Pancake stepper

To power the stepper, I first tried an A4988 driver [20], as this is what many online circuit diagrams used. This did not work as I was initially unaware of how easy the drivers are to break by not including a capacitor or unplugging the stepper while the driver is powered.

I then switched to using an L298N [21] driver, which is not designed for controlling a stepper but is possible by using the Arduino to send signals to each coil manually. The L298N drivers do not have any current limiting capabilities however, meaning that the stepper drew 3A and the driver quickly heated up.

Having learnt from the previous mistakes with the A4988 drivers, I went back to using them. This worked very well, with power consumption decreasing from 40W to 7.2W. I then realised that this was causing the stepper to not have enough torque, so changing the current limiting potentiometer caused the stepper to have a power of 12W.

2.2.4 Arduino

While I am using a raspberry Pi for the webserver and all positioning calculations, I decided against using it directly to power the servos as they require a stable PWM (pulse width modulation) signal. Raspberry Pi's run Linux, which schedules tasks based on priorities but precise pulses may be interrupted by system tasks. This means that the operating system is not real time, resulting in the servo jittering. I therefore decided on using the Arduino to control any actuation, which is sent data via serial.

After switching to 2 MG996R servos I also had the issue that the entire system switched off whenever the arm needed to move upwards. I suspected was that this is to do with the current draw increasing by too much suddenly when all the servos all move at once. To decrease this issue, I added a time delay of 0.25 seconds between when each servo moves, resulting in the maximum current draw decreasing with very little performance impact.

The Arduino code also sends signals to the steppers and calculates how far and which way to move the stepper to reach a particular position. This required a couple of trials, as my initial code just took the shortest route to a position. I then realised that this would not work as just going the most direct route to positions would result in the arm rotating more than 360 degrees, which while faster would cause the cables to become tangled. I fixed this by simplifying the code to just moving forwards if the new position was lower.

2.3 Assembly

2.3.1 3D printed parts

To quickly test parts and work through CAD iterations, rapid prototyping of the designs is required. The method I used for this is 3D printing, as there are far fewer constraints than other methods such as laser cutting. It also has a faster turnaround time and less cost than more industrial techniques such as CNC.

Prusa MK4

I have access to a Prusa MK4 printer [22] (Figure 1.3-1) which can print a variety of filaments well and requires minimal tuning. Given any other printer I use will have a far slower turnaround and is unlikely to have any important features the MK4 does not I decided to not investigate other manufacturing methods unless I had specific material / tolerance requirements.

Prusament PLA

Initially I printed the entire design out of PLA, as it is very easy to print and is reasonably strong. From prior testing, I found that my filament (Prusament) [23] with standard settings needs a tolerance of 0.15mm for a tight fit. Taking this into account, all parts connected well with very few print failures. I later switched to Polymaker PLA when my Prusament spool ran out [24] due to it being significantly cheaper, but there were no dimensional or quality changes to the parts, a common problem with switching to a lower quality filament.

One issue was that the axle of the gear quickly wore down, causing it to break off and lose the connection to the pivot. I then switched from PLA to PETG for the cogs, axle and pivot as it performs better under stress [25]. PETG is also very common and easy to print, and I already had the required calibration settings from using it previously.

After assembling the entire arm, I realised that the axle wearing down was still a problem, even after switching to PETG. There was also the issue that the press fit joint between the cog and axle wore down, causing the design to slip there as well. As good quality filament spools are expensive (£50+ for most specialised filaments) I investigated the different filaments that I already had, which were TPU, Nylon, ABS and ASA. TPU and ABS could be immediately discounted as TPU is a speciality filament for flexible components,

First Nylon tests

Final Nylon tests

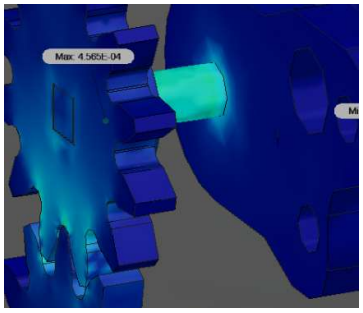
and the ABS was in very bad condition due to its age and exposure to humidity (3D printer filaments must be kept dry to maintain print quality).

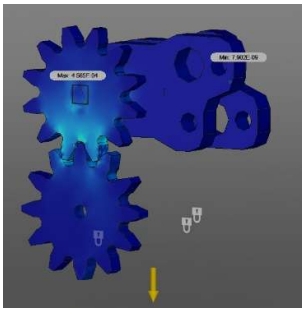
As my initial focus was just on wear, I first researched printing nylon as Prusa's knowledge base suggested it was one of the best for abrasion resistance [26]. However, nylon is notoriously hard to print and the filament I had was unbranded so I did not have any access to datasheets. I therefore made a small CAD model to test print quality and tolerances, which I tested with different settings.

Test No.	Settings	Notes
1	260C nozzle temperature, 0.2mm tolerance, input shaper, PA sheet	Too deformed to test, crackling + discoloured in printing
2	230C nozzle temperature, 0.2mm tolerance, no input shaper, textured sheet with glue stick	Slightly too low tolerance
3	Above with 0.25mm tolerance	Very loose (too high)
4	Above with 0.05 and 0mm tolerance (2 items) and manually lowered speed	Too low
5	Above at normal speed, turned on auto fan, 0.1 and 0.15mm tolerance	Slightly too high
6	0mm tolerance	Works perfectly

I then printed out a joint to test it fully, although I then realised that I had failed to consider the flexibility of nylon, as the axle held together but significantly deformed. After briefly exploring adding software adjustments for this I realized that there was no way to have the joint go fully vertical with this method and calibration would be very complex.

Before continuing further, I decided to do a static stress analysis (a type of finite element analysis) in Fusion [27] to confirm where the deformation is most likely. These images from the analysis (green is more force) continue to show that the axle through the bearing is the weak point.

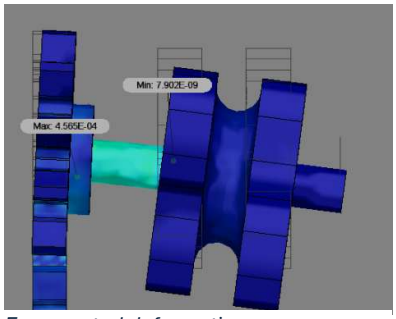




Static stress analysis

More force

Less force



Exaggerated deformation

I then investigated ASA, as this is far more rigid than nylon and is more wear resistant than PETG (although not as much as Nylon) [28]. After going

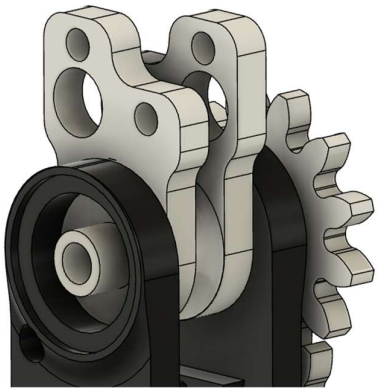
through a similar (although shorter due to being official prusament) process with ASA, I then reprinted the gear and axle with ASA. These gears then lasted significantly longer.

After much more software testing the 3D printed axles began wear down and eventually snap, causing the positioning to become inaccurate as the connection had a lot of give. As ASA has the best properties for this purpose out of filaments I could access, I then considered other non-3D printed materials.

The options for this were laser cut acrylic and metal. I initially researched acrylic, as I had more flexibility on tolerances. Laser cutting it also meant that the axle would be square, meaning that the connection between the axle and gear would be less likely to break. Acrylic and ASA also bind well to superglue.

The issue with laser cut acrylic however is that cutting very small pieces (in this case 4x3 mm) is often unreliable, and the acrylic could snap easily if force is accidentally applied in any other axis. Metal has the disadvantages that I would have to buy a component of the exact size, giving me less freedom on the tolerances. The connection with other parts may be less secure, as metal and superglue bind less well. I therefore decided to use epoxy instead of superglue. The metal axles are also circular, which requires the connection to be even stronger (another reason to use epoxy), although has the benefit that they will not require an extra spacer for inside the bearing.

I decided to try using a screw as the axle, as I had access to a variety of these and the grooves of the thread provide a rough surface for the epoxy to attach to, and this worked well enough that I continued to use this.



Partially modified design

I also later modified the design slightly so there was more contact area between the epoxy and ASA, as small forces caused the epoxy to break. This also meant that there was a lower chance of accidentally gluing the parts of the bearing together as nothing is directly attached to it, making assembly easier. With the increased contact area, I also attempted using superglue as the issue with the epoxy appeared to be more related to curing than actual strength. This worked significantly better than the previous design with epoxy. I also later decided to only use a screw on the gear side, meaning that the back plate can easily be taken off to replace the servo should it be required.

2.3.2 Fastenings

The aim of the fastenings was to allow the design to be quickly and securely assembled and disassembled, as I expected to have to do many iterations on different parts. This meant that the only option I had was bolts or press fit components, as most other

methods (melting nuts into plastic, glue etc) have some permanent post-processing. The disadvantage of press fit components is the importance of tolerances, often meaning the final iteration requires glue. I therefore decided to use bolts for connecting all components.

To make assembly simpler, I also aimed to use screws of the same dimensions in as many places as possible. As most of my design has 4mm thick plastic, and most bolts secure 2 pieces of plastic together, M3x12 (diameter 3mm, 12mm length) bolts fit almost every connection (excluding a few specific holes in pieces of electronics, some of which needed M2). I chose to generally use M3 over M2 due to them being a more common part and their larger size adding stability, decreasing the chances of them coming undone.

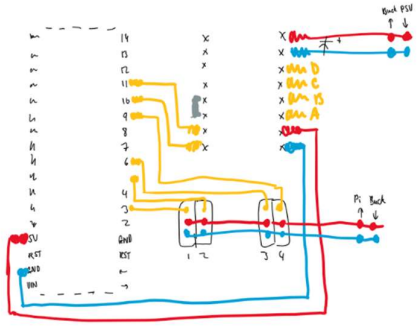
I initially did not use any washers in the design, but after disassembly I noticed that the area around the bolts was slightly inset, which in places had caused the plastic (particularly the ASA) to snap. I added washers to the future joints, which also added stability to the connection due to the distributed load.

I also had the issue that the nuts loosened over time, so I switched to locking nuts [29], initially just on the parts that were shaking loose but later for the entire design as it was far more reliable. While a few of the fastenings did still loosen after a long time, this significantly decreased the issue.

2.3.3 Wiring

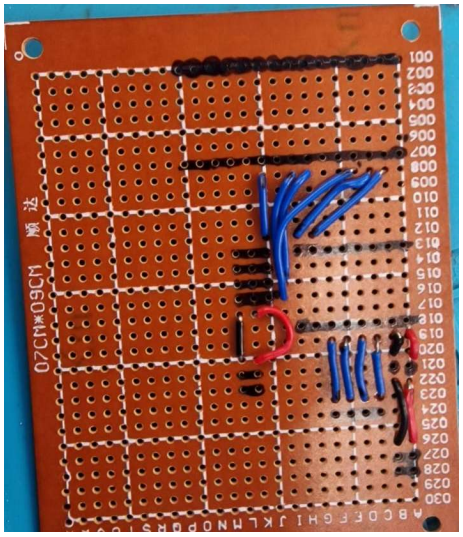
When testing the electronics, the simplest method of wiring was to use a solderless breadboard due to the ease of rewiring the circuit. This would not be suitable for the final design as it is large and wires can become easily disconnected. My options for making a more permanent design were design a PCB (printed circuit board) or use a solder board. A PCB would be better long term as it is more reliable, but I would also need to outsource making it and it did not seem worth it for a project that only aims to produce a prototype.

I decided to use a Veroboard to make the circuit, as I wanted it to have a small footprint. I did not use a stripboard as I would have needed to cut almost all of the tracks. I also used female header pins throughout instead of direct soldering so that I could replace the components easily in case of breakages.



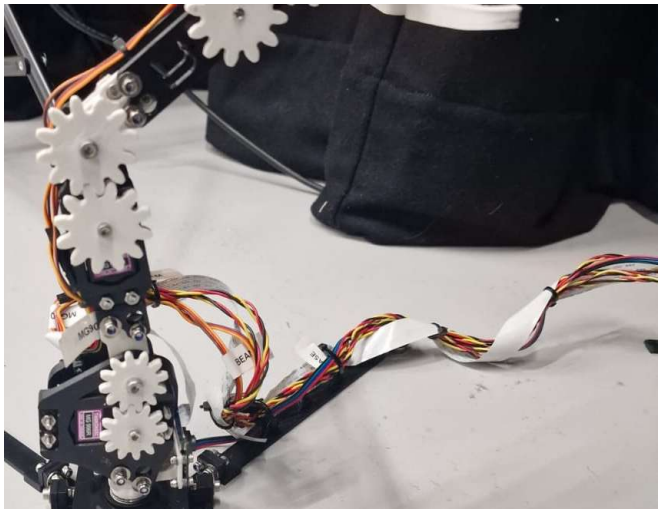
Circuit diagram

Once this was soldered, I then made the connections for off-board cables. As most of these are male to male connectors there was the option of just soldering wires directly to the board and connecting without any crimping. This would make the circuit hard to understand and increase the chance of loose wires falling out, so I decided on crimping each wire [30] and grouping the ones going to the same component in the same connector housing. This made the wires far easier to manage, especially as many were almost half a metre long.



Some of the wires on a Veroboard – positions of header pins drawn on

Relatively late into the project, I remade the circuit with the same wiring diagram but neater to work out if any electrical issues were just due to poor soldering rather than sudden current draw. I also re-crimped most of the wires to exactly the right length, making it neater.



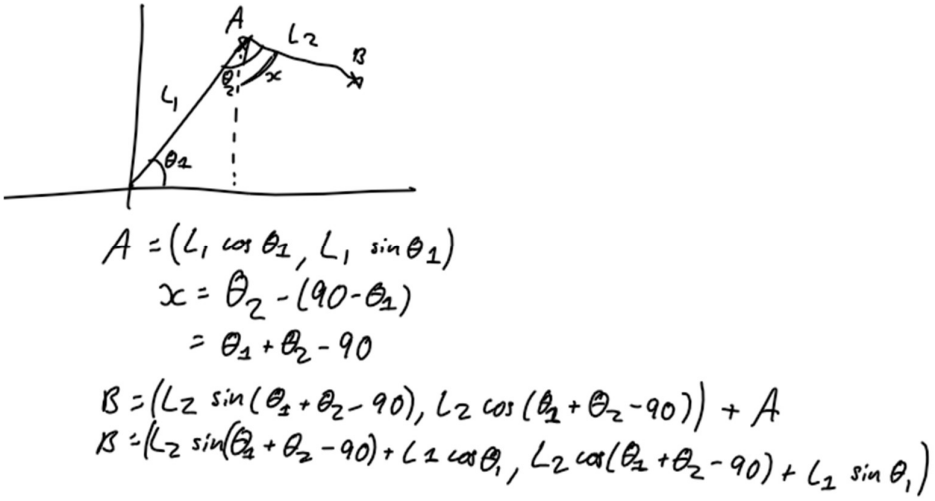
Wiring on the arm

3 Raspberry Pi code

This covers the server-side WebSocket communications, computer vision, script writing and how positions are calculated and sent to the Arduino.

3.1 Positioning

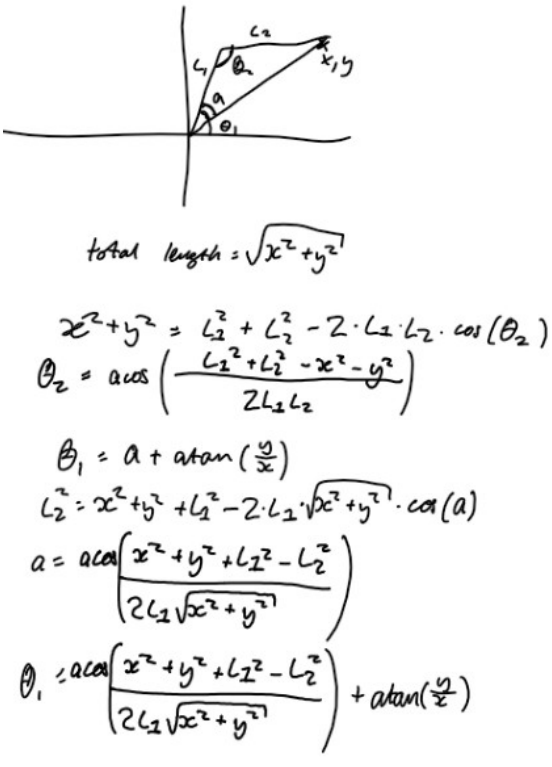
3.1.1 Angle calculations



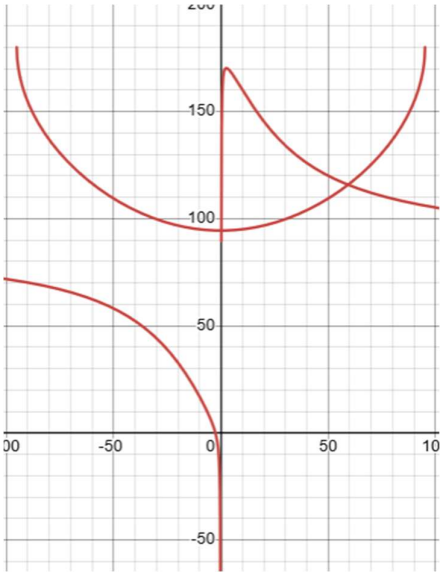
Forward kinematics

The angle calculations are completed using inverse kinematics, which algebraically works backwards from the desired position and lengths of the beams. I first found the formula for these going forwards with two joints, as while I plan to have 3 the third joint will always be parallel to the ground, making it equivalent to two joints getting to (x - [length of joint 3], y).

The next part of this was doing the same process in reverse. After following through a tutorial [31], I re-derived the equations and wrote the code to perform the calculations in Python as the provided code was of very little help due to having never written Go.

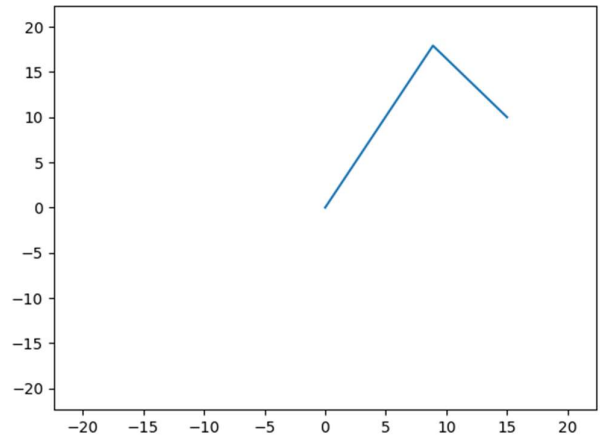


Inverse kinematics for 2 joints



Graph of the joint angles depending on desired x-position

To check these formulas, I then used graphing software to see what values would be possible. With θ_1 and θ_2 on the y-axis and x position on the x-axis (keeping L_1, L_2 and y position constant) this showed that with two 70cm beams at a height of 30cm the arm could reach a maximum of ~95cm in the X direction, which makes sense physically.



Tkinter GUI showing positions of joints to reach a certain point

To further check my calculations, I then plotted the resulting lines on matplotlib graph by drawing the start, end and middle points and connecting them, which looked correct compared to my input beam lengths and other parameters.

3.1.2 Pi-Arduino communications

The most common method of wired communication between a Pi and an Arduino is by using a serial port. I chose to do this using the USB port rather than GPIO as it also provides power to the Arduino and is more reliable. There is also the added benefit of this being the same method to connect the Arduino to a laptop, making debugging easier.

I have previously connected an Arduino and raspberry Pi in another project, so I based the code of an old repository [32]. I reused from this the connection initialisation on both the RPi

```
self.arduino = serial.Serial('/dev/ttyUSB0', baudrate=115200)
```

and Arduino,

```
Serial.begin(115200);
```

how to send data

```
self.arduino.write(bytes("4" + str(claw_pos).zfill(3) + "\n", 'utf-8'))
```

and receive data.

```
while (!Serial.available());
data = Serial.readString();
```

I did not spend time looking at options for communication as I had already researched this relatively extensively when writing the code this was based on. For this reason, this connection had very few issues.

3.1.3 Error handling

While my previous code could reach any position within a ring (inner diameter long beam – short beam, outer diameter long beam + short beam), this is not possible as the joints do not have more than 180 degrees of movement, which is also constrained by the two beams hitting each other, meaning that the angle between two beams cannot be less than around 40 degrees.

I fixed this issue by adding a “tmin” and “tmax” to each beam (a class as part of the arm) and giving each beam a very simple method to check if it is in a valid position.

```
    def possible(self) -> bool:
        return self.tmin <= self.angle <= self.tmax
```

I then called this before sending any data to the Arduino to prevent the arm attempting to turn somewhere impossible, possibly breaking it in the process. This failure was then communicated back to the client in the same way as if the position was out of range.

3.2 Webserver

3.2.1 Websockets / HTML

Initially, I wrote all the code from my laptop, using the serial connection from my laptop's USB port to control the Arduino. This removes any issues with the Pi from my workflow, temporarily simplifying it. I could also temporarily use an extension on VSCode to serve HTML, meaning that I would only need to look at WebSocket communication.

I decided on using the default WebSocket library [33] for Python, as I wanted something relatively simple. In previous projects I have used socketio for a similar purpose, a library that abstracts WebSocket but has previously caused surprising bugs. WebSocket is built of asyncio, so I also used that for the threading of sending data through WebSockets.

The server is started on the Python side:

```
start_server = websockets.serve(echo, "localhost", 8765)
asyncio.get_event_loop().run_until_complete(start_server)
asyncio.get_event_loop().run_forever()
```

And the connection is initialised from the JS:

```
var websocket = new WebSocket("ws://localhost:8765")
```

Messages can be sent and received in Python:

```
await websocket.send(message)
```

```
async def echo(websocket: websockets.WebSocketServerProtocol):
    async for message in websocket:
```

And in JS:

```
websocket.send(message)
```

```
websocket.onmessage = (event) => {}
```

I could not use the local webserver extension on the Pi however, so I then looked into libraries to serve the HTML. Simplicity was also the key consideration for choosing the library, so I went with aiohttp, as it is again based on asyncio.

3.2.2 Connect to laptop's hotspot

Using a raspberry Pi with a screen can often be impractical for debugging, as it is not at all transportable and takes a while to set up. A more practical method of writing code for the Pi is therefore connecting it to a network and SSHing into it.

The most common solution is to connect the Pi to the same network that the laptop is on, but this only works within the network's range and requires SSH to be allowed on the network. Another option is using the Pi as a hotspot and connecting my laptop to the Pi's network. This would mean that I would not be able to access the internet while connected to the Pi, making debugging harder and preventing me from committing anything to GitHub.

The solution that I went with is using my laptop as the hotspot. This has the feature to share another Wi-Fi network as a hotspot, which the Pi can connect to. This provides the Pi with internet but is also transportable as I will have my laptop wherever I want to access the arm. As the Raspbian imager allows you to set a default network when flashing the OS, this means that it is possible to use the Pi without ever connecting it to a keyboard / monitor, although unfortunately debugging connection issues (the Pi uses 2.4GHz but the hotspot defaults to 5GHz) meant this was not the case on the first image I used.

3.3 Objects

3.3.1 Taking Photos

The cameras that I had access to were Picamera2's, so I used the corresponding Picamera2 library [34] for the code. This made the code to take a photo very simple, with the only custom aspect being the camera configuration settings (after some tests with varying exposure, I decided to stick with the default).

3.3.2 Object detection

While I had never used computer vision in a project before, I had attempted to use it for line following over a year ago. I therefore got some of my initial ideas from this (finding contours and location/size information for each contour), but also used a clipping mask for only the green areas.

Aside from finding the correct colour range for each barrel (I tested on green as I already had a green cylindrical object), this was fairly reliable for the test images.

3.3.3 Colour detection

When I was initially testing the arm, I only had access to red and black filament, so all of my barrels were red and I could just mask everything that was the wrong colour. However, I wanted the ability to detect and render barrels of different colours, so I printed some in blue and yellow as well.

My method for doing this was to create masks of each colour individually, find the largest contour across all of them and then return the distance calculated from this plus what colour it was. I first implemented this by just finding the maximum of the three contours, although this then threw errors when some of the contours didn't exist in the image. This only became an issue when I tested in a fully monochrome room, so my initial fix was just adding if-else statements to whether each contour existed, then doing a comparison on the available ones.

This code worked fine, but with 3 barrels quickly became overly long and convoluted, so I switched to having a dictionary that the largest contour for each colour (if applicable) was added to, which significantly shortened the code.

3.3.4 Distance calculation

The distance away should be $\frac{W_a k_1}{W_p}$. From knowing everything aside from k_1 for a range of distances, I tried to find k_1, initially using photos taken with a phone camera. Testing this out I got an increasing value of k_1 as the distance increased, suggesting that the formula is incorrect. After testing different values to get a constant, I found that the formula $\frac{W_a k_1}{{W_p}^{k_2}}$ happened to work perfectly when k_2 is 0.9.

Key:
k_1, k_2: arbitrary constants
W_p: Perceived width
W_a: Actual width

My guess was that this was a result of some focus / optimisation within the iPhone's camera, so I did not research the issue further. This distance formula worked, as it was only 8mm off when I gave it a new image.

I then switched to using the Picamera once it was set up, which required me to find new values for k_1 and k_2. As my previous version was just guesswork as to what was closest, I partially automated the process of finding k_2.

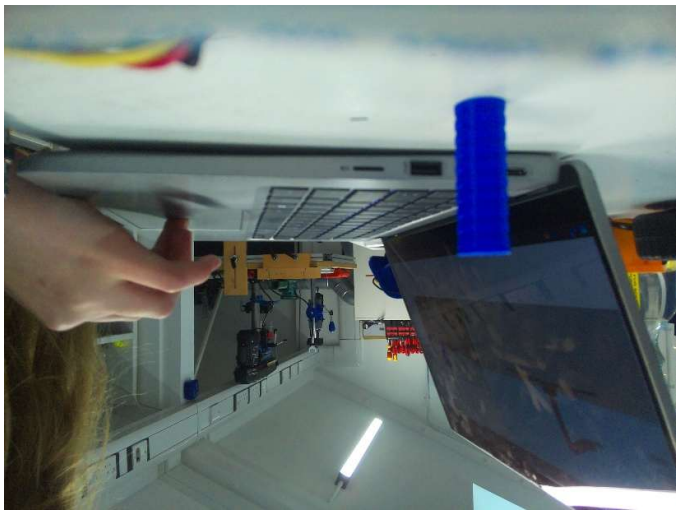
```
print(np.sqrt(np.average([(i[0] * i[1] ** (0.0001 * j)) ** 2 for i in k]) -
np.average([(i[0] * i[1] ** (0.0001 * j)) for i in k]) ** 2) /
np.average([(i[0] * i[1] ** (0.0001 * j)) for i in k]), j * 0.0001)
```

This code prints the standard deviation in k_1 from the value of k_2 and the attempted value of k_2. *j* is just a for loop to decrease the range down by a factor of 100 or so each running (it uses integers that are 000s of times larger than the actual required k_2 to make the range easier, and they are then divided to get an actual value). This process still required a few runs and some manual intervention, so I considered writing a very basic regression algorithm to speed this up. I decided against it however, as it was not something I planned on repeatedly doing.

Picamera taking picture of test barrel

After substituting the calculated values of k_1 and k_2, the distance calculations were still very accurate (5mm tolerance).

All of this testing had been using test barrels, so I then switched to using the actual barrels. By this time I also had a proper mount for the camera, and I discovered that the issue I was having before with the unknown constant k_2 was no longer the case. This simplified the process of finding k_1 significantly, and also made more logical sense, as presumably k_2 was just from random error that happened to follow a trend. My code to programmatically find the distance then used this calculated k_1.



Picamera taking a picture of a real barrel

3.3.5 Scanning for objects

To find where objects were, I first needed to convert the distance from the camera to the barrel into the x-position the arm needed to be set to so that it could pick up a barrel, as there was likely some offset. I therefore tested the required versus calculated distance across a range of values.

Perceived distance	Arm distance
112	110
118	120
128	130
139	140
149	150

As it happens, no adjustment for the offset was required, although this is just due to how the design turned out. Another aspect of the distance code that I needed to consider was that the barrel may not be completely centred on the camera, which would cause detected position to be incorrect. I therefore wrote a function to find how far off (in pixels) the object is from where it is supposed to be and in what direction.

This finds the x coordinate difference between the centre of the whole image and centre of the contour. If there is no contour then it just returns a large number, causing any code to attempt to centre using this to rotate significantly, meaning it looks elsewhere.

If the centres are within 100px of each other, the arm considers itself to have centred on the object, causing it to add a barrel to its list and turn significantly so that it does not return to the previous barrel. It checks at various points if it has moved over 360

degrees. Otherwise, it moves proportional to how off centre the barrel currently is. I did not spend much time calibrating this, as it is fine if it slightly over/under steers as it will still centre on the object after a few steps.

When each barrel is found, it is added to a list (so it can be found for pickup), and data about it is sent to the client. The barrel class is very simple, with the only method formatting some of its attributes to send over websocket. This uses cartesian rather than the server's polar coordinate system, so each barrel automatically converts between the two despite the cartesian coordinates being irrelevant on the server side.

After some of my joint / claw length modifications I had to add an offset of -40mm to consider the increased length and previously inaccurate measurements, calculated through a similar process to the table above.

3.3.6 Picking up objects

This function does not require finding barrels initially, as they have already been detected through a scan. This means that my initial code for this just involved moving up to a "resting position" (so it did not get in the way of anything on the ground), turning and reaching down to the stored position.

After testing this, I then realised that another issue was the claw hitting the barrel and knocking it over when it reaches down. To fix this, the arm instead first moves all the way to the ground and as close as possible to itself, then moving slightly up and out to the barrel itself.

Another feature that I added to this was checking if the barrel was picked up. This checks if there was still a barrel in the position, and if so, repeats the whole process of attempting to pick it up. If not, it modifies the barrels status to being gripped by the arm, and notifies the client of the status change to update the render.

3.4 User defined scripts

3.4.1 Running code

My initial thought on how to do this was to write my own basic parser. However, after researching how to do this [35], I realized that writing one properly was an entire project in itself. I therefore switched to simply allowing the user to execute their own Python code. This had a few disadvantages, most notably the lack of security as anyone with access to the webserver could hypothetically inject code to wipe the Pi. As this only runs on a local network it was not an issue as only devices with my laptop's Wi-Fi password would be able to run code

When the server receives code to execute, it is run using Python's built-in exec command [36]. To allow access to other relevant parts of my code, I also passed in the functions imported from other files to the exec function via "gobals()".

This came with the issue that these functions (scan(), pickup() etc) require variables to be passed in that are common to the entire program, such as an instance of the arm class or the websocket. It would be perfectly possible to just pass these in in wherever required, but it would make the user's job harder.

To fix this, I wrote lambdas [37] for each function that take in any number of arguments and then run the function, passing in the first parameters before the ones that were passed into the lambda.

```
move = lambda *args: movement.move(arm, websocket, *args)
```

As these are defined within the parse function, "locals()" is now passed into the exec with the code.

3.4.2 Code outputs

Another important aspect to being able to write scripts is the ability to see what they are doing and if they had successfully completed. Without access to a proper debugger, the easy-to-use method is console outputs. However, with my current design, print statements and error outputs are only visible on the server's terminal, which is inaccessible to the website.

To replace this, I created my own output function, which the user could call to add debug statements, which would be sent to the client live.

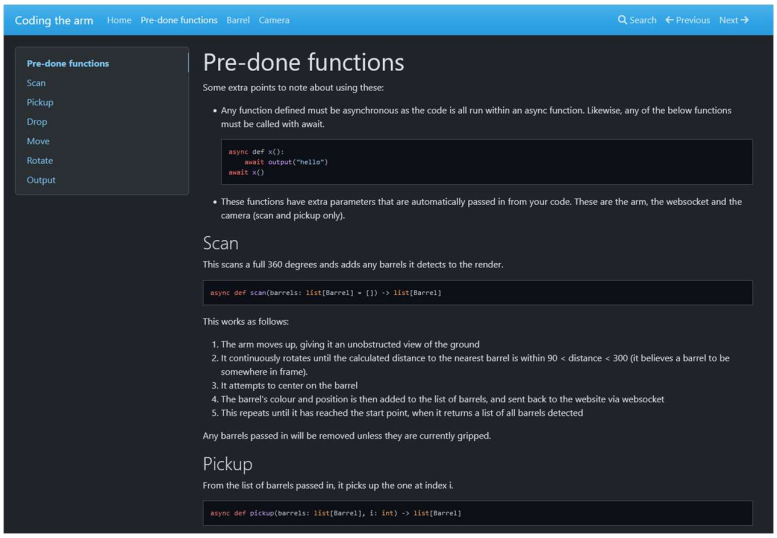
```
async def output(x: str) -> None:
    await websocket.send("output "+x)
```

I also surrounded the executed code in a try-except block, which outputted any errors that occurred during runtime.

3.4.3 Documentation

To make it possible to write code for the arm without having full knowledge of the source code, I wrote documentation for the functions exposed to the user. My aim was for this to have a professional appearance but not have to spend any time ensuring it looked nice, so I decided to use a tool that did all the formatting for me.

A free, popular tool that I found was MkDocs [38], which uses a YAML configuration file and multiple markdown pages to easily produce professional-looking documentation. After working through the tutorials on the MkDocs site, I wrote my own markdown to document the functions. MkDocs then allows you to temporarily host it for testing and then compile to JS.



Generated documentation

One issue I faced with this was that the compile process used folders to navigate to different pages in the documentation, presuming the webserver would interpret that as visiting index.html in that directory. Unfortunately, the webserver I was using did not allow this without writing my own redirect function.

MkDocs allows custom JS to be added to the website, so I solved this by writing a very short JS program that added “index.html” to the end of any link to an HTTP address, as any external links were encrypted so used HTTPS. This replaced any links to other parts of the documentation with working equivalents as soon as the initial JavaScript ran.

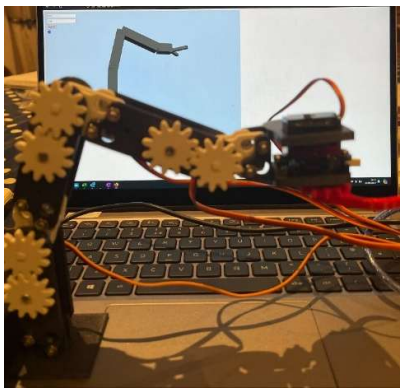
4 Client code

4.1 WebGL

4.1.1 Beam rendering

Prior to this project, I had used WebGL once before, in writing a 3D online multiplayer game [39]. This meant that I already had an example to base the initial code on, so rendering shapes initially was not too hard.

I sent the current position of the arm as an array, with each item containing the centre of the beam, its rotation relative to the ground and the length of the beam. This was encoded and decoded by Python / JavaScript's respective JSON libraries. I then scaled, translated and rotated each beam to the required position.



Early iteration of the hardware and basic arm render

The user needed to always be centred on the render, but I wanted to allow them the ability to rotate around the arm. While I did not plan to implement this yet, I still needed to find a camera position so I could test that the beams were in the correct place.

After initially attempting this manually, to little success, I decided to use LookAt, a built-in function [40] that translates / rotates the matrix so it is pointed at the origin, but at the point specified.

```
mat4.lookAt(modelViewMatrix, [xpos, ypos, zpos], [0, 0, 0], [0, 1, 0]);
```

After getting this working, I implemented allowing the user to rotate. My initial idea was to find an initial mouse position on mouse down, a final on mouse up and just use the difference between them to determine rotation by converting x/y rotation into 3D coordinate a constant away from the centre.

However, I quickly realised that while this worked fine, it did not have the nicest user experience as the rotation was only visible after the action had been completed. To improve this, I modified the code to have a starting camera rotation, which if the mouse is down is updated to the difference the mouse has moved plus the initial rotation. The mouse's position is stored each time mouse movement is detected.

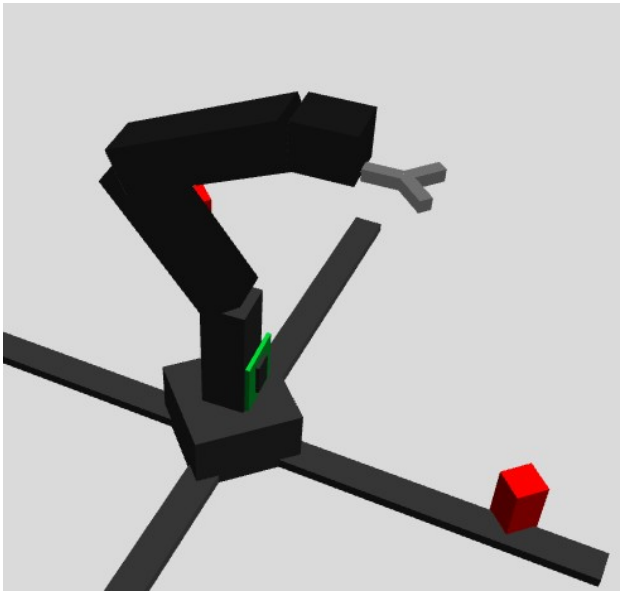
4.1.2 Other arm rendering

To render the beam, I was using a single cube buffer [41] and just modifying it. While likely not the best strategy to make a perfect render, I continued to do this for the claw.

Initially, I just had two rectangles either pointing outwards or straight ahead, although during a UI upgrade I decided to improve this slightly.

While the majority of the design was not sticking to the colour scheme of the arm as individually rendering the pivots and gears would be a large amount of unnecessary effort, as the claw was entirely ASA I did load white texture to differentiate it from a beam. I also rendered the claw symmetrically, as there were a relatively large number of error-prone transformations that I did not want to have to rewrite.

I first translated the nearer part of the claw to its centre point, which was not on the same X/Y plane as the rest of the beam due to it being angled. This was 45 degrees in the closed position, and 90 degrees in open. After having scaled and drawn this, I reverted back to the initial matrix (to remove the transformations) and moved to the end of the first part plus the middle of the next one (angled inwards by 45 degrees from the earlier part). I then did similar scaling and rotating transformations to draw this half of the claw as well.



Debugging rendering the claw

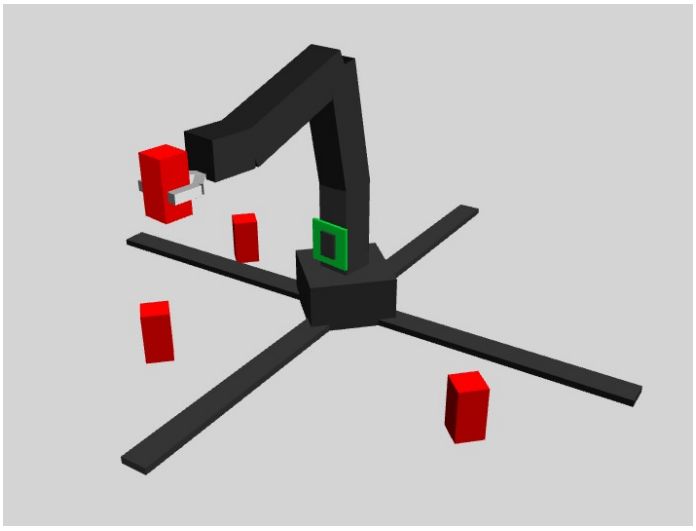
While only required for a 90-degree and 45-degree open claw, this code would work for any angle that the claw could be at. I added this functionality in case I wanted to give the user more freedom on the claw position, although I later realised that this would be of no use.

My aim of the render was to provide a stylized, understandable representation of the arm without going into unnecessary detail. To achieve this, I added green and black rectangles on the front of the arm, representing the Picamera and giving the user an idea of where it was pointing. I also added a base, floor and the stabilising beams extending out from the base to give a better idea of scale and to make the render slightly more realistic.

I also rotated the whole matrix before rendering the beams, meaning that any received rotation would be immediately noticeable in comparison to both the viewport and the base / any barrels.

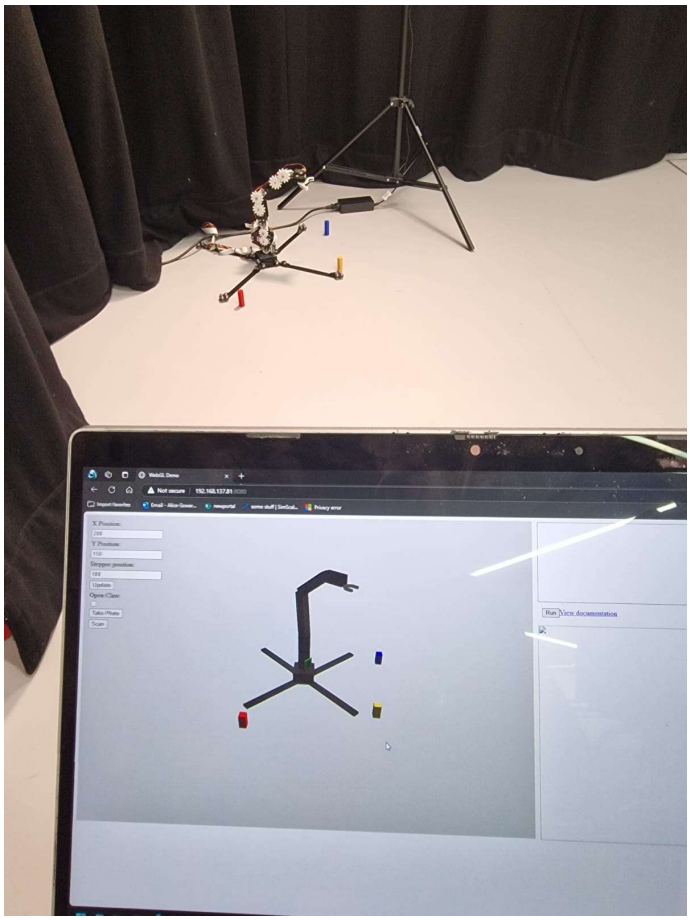
4.1.3 Barrel rendering

Each barrel's coordinates and a string describing the colour are sent via WebSocket, making rendering easy as there was no rotation to consider. I decided not to attempt to make this circular, as so far I had managed with only cubes and did not think it was worth adding a second buffer just for this.



Many barrels rendered, one of which has been picked up

The complex part of the barrels was sending the server a request to pick up a barrel if it is clicked. I did initially consider ray casting [42], where I would draw a line through all the points in 3D space that the click goes through. This looked hard to debug and more complicated than I had time to a implement within the scope of this project, so I decided against doing this.



Positions of barrels, also displayed on render

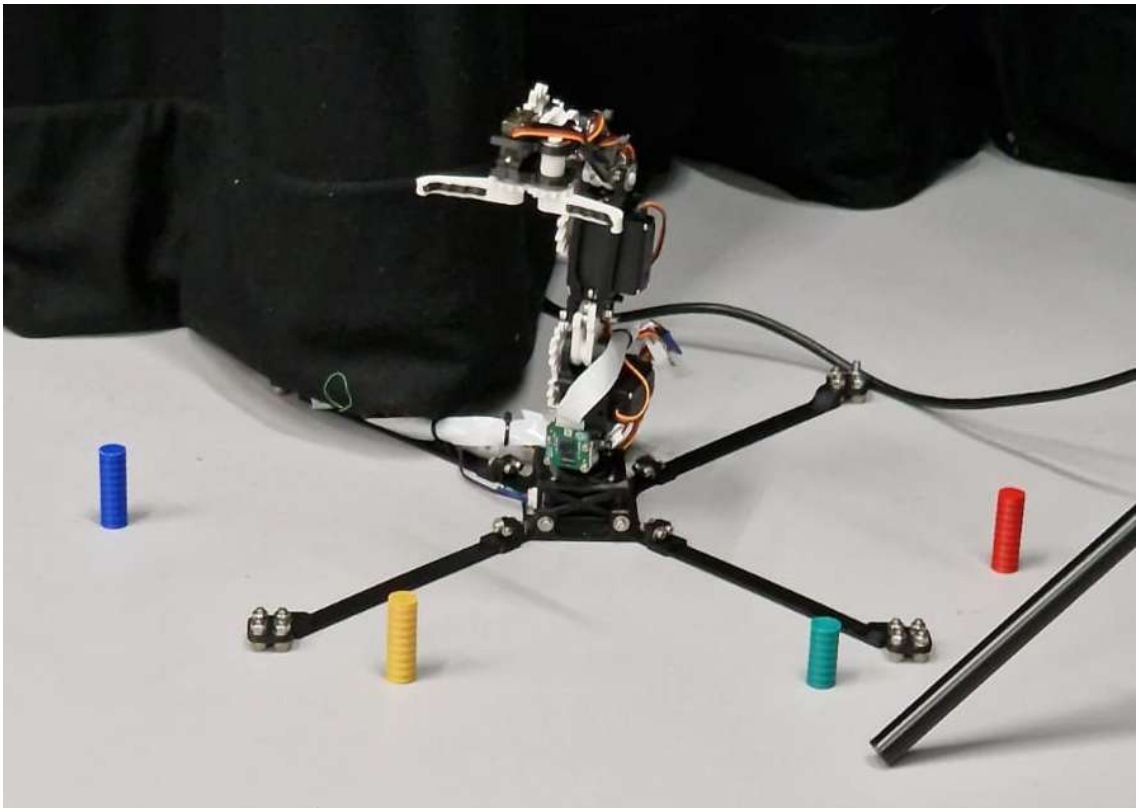
A common method of doing this [43] is drawing the scene on an offscreen canvas with each object a different colour, and using the colour at the point clicked on this canvas to work out if anything was clicked, and if so what.

To draw this offscreen scene, I reused most of the same code as before, but only rendering barrels. When adding a barrel to the list in the client, I gave each one a colour ID of a shade between red and

yellow. These all have a red value of 255, meaning that the green value can be calculated even in shadow by using a map.

From this, the index of the barrel (if applicable) can be measured by dividing the green value by 10.

Barrels that are picked up are not rendered in this first pass but are instead rendered at the end of the arm.



Renders and physical positions of the arm

4.2 Other UI features

4.2.1 Arm control input

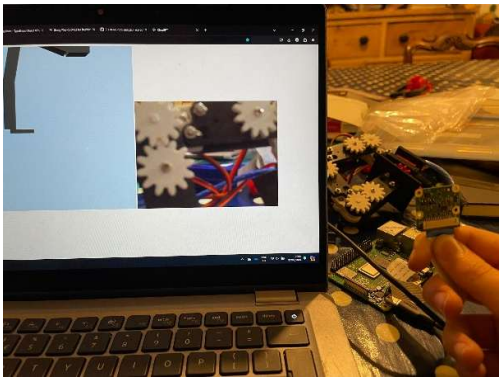
Aside from clicking on barrels, I also allowed the user to set the coordinates of the arm. This is via 3 input fields, X, Y and stepper position. When they press update, it sends the request to the server, which either updates the arm and render or notifies the user that there was an error in the request. This is done by a small red text overlay on the render. There is also a checkbox to open or close the claw, and a button to initiate scanning.

Main input controls

4.2.2 Image display

I also wanted to allow the user to take a photo to get an idea of what the camera could see. My initial thought on how to do this was to use the capture array [34] function that I was already using for computer vision. This produced a NumPy array, so I serialised it via JSON before sending it over WebSocket. On the JavaScript side, I then drew a canvas and filled each pixel with the RGB value from the array.

This was a very slow and inefficient approach, as the JSON string was significantly larger than the initial image due to how it was encoded. Drawing each pixel was also a lot of effort for the client, and I wanted to minimise this as much as possible as it was already rendering the whole arm every frame.



Picamera taking a photo, sent through websocket

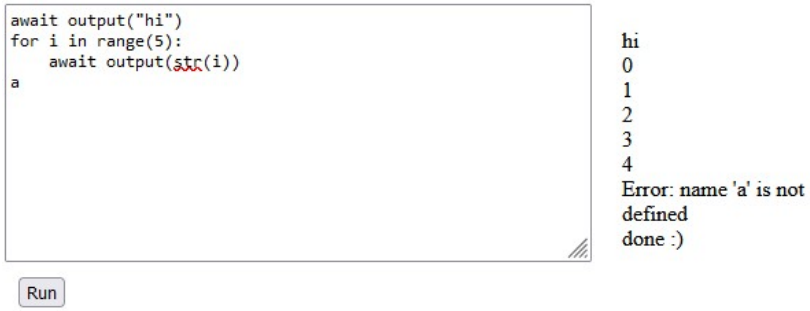
I therefore changed my approach entirely to saving the current position as a JPG in the static file, then just sending a notification that it had updated over websocket. As the image is sent directly over HTTP and is compressed, this made loading significantly faster. I also loaded the image through changing the content of an HTML image tag, which takes significantly less processing.

One issue with this was that the browser attempted to cache the image rather than reload it. This meant that despite the JavaScript resetting the content, the image did not update. To fix this I added a parameter of a random number (1-1000) to the end of the request source. This makes no difference to the image loaded, but means that the cached one is not used instead.

I attempted to also update the image during scans, so the user could see the barrels as they were detected. However, these images did not load due to the file constantly being rewritten, making it inaccessible as it was being modified. As the obvious solution was to slow down the scan so that there was a longer window where the image could be loaded and I considered speed to be more important, I did not implement this.

4.2.3 Script UI

The third part of the screen is the UI for writing scripts. This has a large textbox to write code in, with a run button and a hyperlink to the documentation that opens in a new tab so the website is not refreshed. When output is called in the running code, it also updates the tab to the side with the requested string. This is then cleared when the code is re-run.



Script writing UI (excluding documentation hyperlink)

5 Evaluation

5.1 Mid-project review

This was written w.c. 14/02/2025

I have made quite a bit of progress overall on this project, although not always in the areas that my Gantt chart said I would. More of this progress has been on the hardware than software side, as I realised that I needed the hardware to be relatively reliable before I could begin to work on the software.

The process of designing the joints has gone well so far, although I very much doubt my current iteration will be the final version. The gear mechanisms and connectors between joints are a solid basis for any future iterations, which will likely focus on improving strength and reliability. Having easy access to a 3D printer and knowledge of CAD software has helped a great deal in speeding up this process, although my current prototyping workflow still needs further streamlining. Hopefully, the process of tolerance / print settings testing that I did last week can be applied to other materials, and I can use the techniques I have learnt so far for modularisation and design for manufacture to decrease assembly time. Choosing and connecting electronics has also been more time consuming than initially planned, although I hope that this is now near completion.

Due to the low proportion of my time that I have spent on software rather than hardware, I have been less successful in making progress on it. All of what I have written appears to work well and hopefully will not need too many modifications to add my other planned features. The next step is to get the camera set up and start computer vision, which will require a lot of time spend calibrating / debugging. This may become a time management issue for the second half of the project, as a lot of the script writing will require very reliable hardware and computer vision, which may be challenging to do before the deadline.

5.2 Final evaluation

The result of the design side of the project met all the requirements and was reliable enough to work with computer vision. What was unexpected with this was where the difficulty was: I was presuming that designing a theoretical reliable part would be the hard aspect, although towards the middle of the project I realised that material choices and how to design around this was a lot of what I needed to work on. With previous projects, I have generally stopped at the “just works” stage, so I never came across a lot of these precision issues. Through this project, I have now learned techniques to improve reliability in hardware and make my designs more maintainable in the long term, which hopefully I can incorporate into my initial designs for future projects.

Having initially considered the electronics to be a small aspect, I was surprised by how long choosing the correct components took. As with the design, it was relatively easy to make a quick protype that met most of the requirements but finding ways to produce a reliable device took significantly longer. The part of this that I found most interesting was figuring out power management for the stepper, as this led to research on boost converters which, while irrelevant for this scope due to complexity / reliability issues, was still a very interesting topic from an electrical engineering perspective.

A reasonable portion of time spent working on the hardware required very little skill, as it was just reassembling different parts to replace components or repetitive soldering / wire crimping / fastening. This time was not wasted, as I did focus on designing for assembly from the start, and with any project with lots of parts assembling it from scratch will take time. The process of doing these did speed up significantly as I found better assembly methods or modified the design, but it was still not a particularly fun aspect of the project.

Writing the code was relatively predictable, as I had used a few similar technologies before and knew the rough difficulties of the major elements. However, it was still very interesting to write and covered many things that I had not used before. Towards the end, I realised that my planned methods for implementing picking barrels and running user-defined code would be too complex for this project's timescales, so I found viable but less computationally interesting methods, meaning that the final product still meets all the project objectives.

What was particularly enjoyable on the software side of the project was the number of different technologies used. The code incorporates physical connections with many different types of processing, a webserver with WebSocket connection and a client that uses low-level WebGL libraries. The whole system of requesting a position update from the browser, it being sent to the server, the server calculating how to reach that position and then sending the Arduino a signal to do so and updating the client with the position which the client then renders was hard to debug but extremely rewarding when it worked.

An aspect of the project that took a large amount of time but is not very visible in this document is the calibration / final debugging stage. When working with so many different pieces of code, it was very hard to figure out what aspect the problem related to, before I could even attempt to debug it. This was especially an issue with picking up barrels, as the hardware, client click detection and computer vision distance calculations all needed to be very accurate for it to work reliably. I did not anticipate quite how long this would take, so I ended up having to spend a lot of time in the last few weeks testing and making small changes to hardware and software. Despite these issues, the result is reliable and successfully meets all my project objectives to a good standard.

5.3 Does it meet the objectives?

5.3.1 Hardware / lower-level code

With a small amount of initial calibration, the arm stays reliable for at least 30 minutes (1a). It can pick up objects using the claw (1b, c) and has 360 degrees of freedom at the base. Some of the joints only have 135 (rather than 180) degrees of freedom, although this is due to discovering it is not required, implying that the objective is not worded correctly rather than a flaw in the design (1d).

Due to the previously mentioned physical characteristics, the arm is available to reach a range of positions with the claw parallel to the ground (2a). All the motor positions are controlled with absolute values aside from the stepper, which the code continuously keeps track of (2b), meaning the arm's position is always known. I have also prevented the arm from moving to a position outside the degrees of movement (2c).

5.3.2 Client

Despite being quite abstracted, the render still clearly represents the arm and provides a good, accurate depiction of its current position (3a). This display can be rotated, with different positioning controls (3b) and any detected objects are rendered relative to the arm's position (3c). The user can also click on these objects to pick them up (3d), and any user input is checked for viability, with an error displayed if the action is not possible (3e).

Images can also be requested from the Pi from the user interface (4a), which are displayed on the client's screen (4b). There is also a box for the user to write their own scripts (5a), with the ability to print outputs while the code is running and documentation for the functions that they can call (5b). These scripts can also be run on the arm (5c).

5.3.3 Server

The server can receive and act upon multiple types of WebSocket commands, including image requests (6a), position commands (6b), and information about detected objects (6c). The computer vision can detect the barrels and find their location with a tolerance of ±5mm (7a), better than required in the objective. While I do not detect moved objects, I record their new position and check if picking up an object is successful (7b).

User defined scripts can be run (8a), and any errors in them are handled and output without crashing the program (8c). Documentation and print statements are debugging options for the user (8b).

Both a WebSocket connection and HTML are served (9a), and the arm is moved based on WebSocket commands (9b). The Picamera also takes images when required (10a).

5.4 Future / what I would do next time

One aspect of this project that I would like to improve should I take this further would be to redo some of the areas in better, more interesting ways. This would include doing some of the software features in the way that I initially planned and designing the electronics in a more low-level way, possibly designing my own PCB that also incorporated a voltage regulator and stepper driver.

There are other features that I would have also liked to add given additional time beyond the project scope. These include using a stereo camera system to be able to detect objects of different sizes and possibly adding wheels to the base.

The accuracy of the arm is what I could improve most in a more realistic timeframe, although through the iterations it has already significantly improved. I could likely use belts rather than gears and make components out of more robust materials to do this, which would be more expensive but more precise. This would also require redesigning the entire arm and a significant portion of the electronics, so was not feasible given the time constraints.

I am pleased with the result of this project, so am happy with most of the earlier decisions that I made. The main thing that I would change is to focus more on designing for assembly from the very beginning, as this was where I lost the most time to repetitive tasks. There were also aspects of the hardware that I could have researched further before attempting a solution through trial and error, possibly speeding up the process. However, I am unsure as to how much this would have helped as attempting to implement this research would also have required many iterations, possibly not making it any easier.

6 Bibliography

[1] S. McGarry, “ An Overview of Parametric Modeling in Fusion 360,” [Online]. Available: https://www.autodesk.com/products/fusion-360/blog/parametric-modeling-fusion-360-overview/.

[2] The Pi Hut, “What's the Difference Between DC, Servo & Stepper Motors?,” [Online]. Available: https://thepihut.com/blogs/raspberry-pi-tutorials/whats-the-difference-between-dc-servo-amp-stepper-motors. [Accessed 1 December 2025].

[3] Arduino, “Nano,” [Online]. Available: https://docs.arduino.cc/hardware/nano/. [Accessed 5 October 2024].

[4] Raspberry Pi, “Raspberry Pi product series explained,” [Online]. Available: https://www.raspberrypi.com/news/raspberry-pi-product-series-explained/. [Accessed 5 10 2024].

[5] Mozilla, “JavaScript data types and data structures,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures. [Accessed 1 September 2024].

[6] TypeScript, “What is TypeScript?,” [Online]. Available: https://www.typescriptlang.org/. [Accessed 15 March 2025].

[7] Three.js, “What is WebGL and why use Three.js,” [Online]. Available: https://threejs-journey.com/lessons/what-is-webgl-and-why-use-three-js#. [Accessed 6 November 2024].

[8] R. McGillivray, “Thrust Bearings - An Overview,” [Online]. Available: https://tameson.co.uk/pages/thrust-bearings. [Accessed 6 October 2024].

[9] Electronicos Caldas, “MG90S Metal Gear Servo,” [Online]. Available: https://www.electronicoscaldas.com/datasheet/MG90S_Tower-Pro.pdf. [Accessed 27 September 2024].

[10] JSumo, “SG90 Micro Servo Motor,” [Online]. Available: https://www.jsumo.com/sg90-micro-servo-motor. [Accessed 27 September 2024].

[11] Electronicos Caldas, “MG996R High Torque Metal Gear Dual Ball Bearing Servo,” [Online]. Available: https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf. [Accessed 10 October 1].

[12] Raspberry Pi, “Power Supplies,” [Online]. Available: https://github.com/raspberrypi/documentation/blob/develop/documentation/asciidoc/computers/raspberry-pi/power-supplies.adoc. [Accessed 1 December 2024].

[13] Amazon, “Miady 14.8V 99Wh V Mount/Lock Battery Compatible with Sony Camcorder,” [Online]. Available: https://www.amazon.co.uk/Miady-Compatible-Camcorder-Broadcast-Replacement/dp/B08PD8NKZW. [Accessed 30 November 2024].

[14] Tenergy, “LiPo Safety Warnings,” [Online]. Available: https://power.tenergy.com/lipo-safety-warnings/. [Accessed 3 December 2024].

[15] The Battery Shop, “Y2.1-12 Yuasa Yucel Yuvolt 12v 2.1Ah Lead Acid Rechargeable Battery,” [Online]. Available: https://www.thebatteryshop.co.uk/y21-12-yuasa-yucel-yuvolt-12v-21ah-lead-acid-rechargeable-battery-598-p.asp. [Accessed 7 December 2024].

[16] Amazon, “TECNOIOT 35W DC 5.5-30V to 0.5-30V LCD Display Step Up Down Buck Boost Converter,” [Online]. Available: https://www.amazon.co.uk/TECNOIOT-5-5-30V-0-5-30V-Display-Converter/dp/B0B6SQ1S1N. [Accessed 5 December 2024].

[17] Kiatronics, “28BYJ-48 5V Stepper Motor,” [Online]. Available: https://www.mouser.com/datasheet/2/758/stepd-01-data-sheet-1143075.pdf. [Accessed 30 November 2024].

[18] Oriental motor, “Stepper motor frame sizes,” [Online]. Available: https://www.orientalmotor.com/stepper-motors/stepper-motor-frame-sizes.html. [Accessed 4 December 2024].

[19] Amazon, “GERUI Stepper Motor 1A 23 mm Body Length 17 Ncm,” [Online]. Available: https://www.amazon.co.uk/GERUI-Stepper-Retains-Resistance-Pancake/dp/B0DCB6DH8Z. [Accessed 21 December 2024].

[20] Pololu, “A4988 Stepper Motor Driver Carrier,” [Online]. Available: https://www.pololu.com/product/1182. [Accessed 28 January 2025].

[21] Last Minute Engineers, “Control Stepper Motor with L298N Motor Driver & Arduino,” [Online]. Available: https://lastminuteengineers.com/stepper-motor-l298n-arduino-tutorial/. [Accessed 27 January 2025].

[22] T. Hoffman, “Original Prusa Mk4,” [Online]. Available: https://uk.pcmag.com/3d-printers/152092/original-prusa-mk4. [Accessed 1 October 2024].

[23] Prusa Research, “Prusament Galaxy Black PLA,” [Online]. Available: https://www.prusa3d.com/product/prusament-pla-prusa-galaxy-black-1kg/. [Accessed 7 October 2024].

[24] Polymaker, “PolyLite PLA Pro,” [Online]. Available: https://polymaker.com/product/polylite-pla-pro/. [Accessed 3 March 2025].

[25] Prusa Research, “Prusa Material table,” [Online]. Available: https://help.prusa3d.com/materials. [Accessed 17 November 2024].

[26] Prusa Research, “Polyamide (Nylon),” [Online]. Available: https://help.prusa3d.com/article/polyamide-nylon_167188. [Accessed 7 February 2025].

[27] Autodesk, “Static Stress Study,” [Online]. Available: https://help.autodesk.com/view/fusion360/ENU/?guid=SIM-STATIC-STRESS-ANALYSIS. [Accessed 3 February 2025].

[28] Prusa Research, “ASA,” [Online]. Available: https://help.prusa3d.com/article/asa_1809. [Accessed 15 February 2025].

[29] RS Components, “A Complete Guide to Locking Nuts,” [Online]. Available: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/locking-nuts-guide. [Accessed 23 November 1].

[30] Wirefy, “A Beginner's Guide to Wire Crimping Tools: How to Choose and Use Them Effectively,” [Online]. Available: https://wirefyshop.com/blogs/news/a-beginners-guide-to-wire-crimping-tools-how-to-choose-and-use-them-effectively. [Accessed 25 December 1].

[31] Applied Go, “Inverse Kinematics - How to move a robotic arm,” [Online]. Available: https://appliedgo.net/roboticarm/. [Accessed 16 October 2024].

[32] UPDATE THIS, “Piwars-2024-2 - GitHub,” [Online]. Available: https://github.com/ice-cube-1/piwars-2024-2. [Accessed 23 September 2024].

[33] Websockets, “About,” [Online]. Available: https://websockets.readthedocs.io/en/stable/. [Accessed 4 January 2025].

[34] Raspberry Pi, “Picamera2 manual,” [Online]. Available: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf. [Accessed 23 February 2025].

[35] S. Setunga, “Writing a Parser - Part I,” [Online]. Available: https://supunsetunga.medium.com/writing-a-parser-getting-started-44ba70bb6cc9. [Accessed 20 01 1].

[36] Python3 documentation, “Builtin functions,” [Online]. Available: https://docs.python.org/3/library/functions.html#exec. [Accessed 12 January 1].

[37] Python3 Documentation, “Lambda expressions - Control Flow,” [Online]. Available: https://docs.python.org/3/tutorial/controlflow.html#lambda-expressions. [Accessed 28 February 1].

[38] MkDocs, “Getting Started,” [Online]. Available: https://www.mkdocs.org/getting-started/. [Accessed 12 January 1].

[39] A. Goward, “3D game frontend - GitHub,” [Online]. Available: https://github.com/3d-game-frontend. [Accessed 2024 December 21].

[40] GL Matrix, “LookAt - Mat4,” [Online]. Available: https://glmatrix.net/docs/module-mat4.html. [Accessed 24 December 2024].

[41] Mozilla, “Creating 3D objects using WebGL,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Tutorial/Creating_3D_objects_using_WebGL. [Accessed 26 December 2024].

[42] Adobe, “What is raycasting?,” [Online]. Available: https://www.adobe.com/uk/products/substance3d/discover/what-is-ray-casting.html. [Accessed 4 June 1].

[43] WebGL fundamentals, “Picking,” [Online]. Available: https://webglfundamentals.org/webgl/lessons/webgl-picking.html. [Accessed 12 March 1].

7 Appendix: Full codebase

7.1 Client source code

7.1.1 Typescript

7.1.1.1 draw-scene.ts

```
import {ProgramInfo, Barrel} from "./main.js";
import {Buffers} from "./cube-buffer.js"

function drawScene(gl: WebGLRenderingContext, programInfo: ProgramInfo,
buffers: Buffers, cameraRotationX: number, cameraRotationY: number, positions:
number[][], zoom: number, angle: number, barrels: Barrel[], stepperpos:
number) {
    var xpos =-zoom * Math.sin(cameraRotationX) * Math.cos(cameraRotationY);
    var ypos = zoom * Math.sin(cameraRotationY);
    var zpos = zoom * Math.cos(cameraRotationX) * Math.cos(cameraRotationY);
    gl.clearColor(0.8, 0.9, 1.0, 1.0);
    gl.clearDepth(1.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    const fieldOfView = (45 * Math.PI) / 180;
    const canvas = gl.canvas as HTMLCanvasElement;
    const aspect = canvas.clientWidth / canvas.clientHeight;
    const zNear = 0.1;
    const zFar = 5000.0;
    const projectionMatrix = mat4.create();
    mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
    var modelViewMatrix = mat4.create();
    mat4.lookAt(modelViewMatrix, [xpos,ypos,zpos], [0,0,0], [0,1,0]);
    const normalMatrix = mat4.create();
    mat4.invert(normalMatrix, modelViewMatrix);
    mat4.transpose(normalMatrix, normalMatrix);
    setPositionAttribute(gl, buffers, programInfo);
    setTextureAttribute(gl, buffers, programInfo);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, buffers.indices);
    setNormalAttribute(gl, buffers, programInfo);
    gl.useProgram(programInfo.program);
    gl.uniformMatrix4fv(programInfo.uniformLocations.projectionMatrix, false,
projectionMatrix);
    gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false,
modelViewMatrix);
    gl.uniformMatrix4fv(programInfo.uniformLocations.normalMatrix, false,
normalMatrix);
    gl.activeTexture(gl.TEXTURE0);
    gl.uniform1i(programInfo.uniformLocations.uSampler, 0);
    const vertexCount = 36
```

```
    const type = gl.UNSIGNED_SHORT;
    const offset = 0;
    const realinitialmatrix = mat4.clone(modelViewMatrix);
    var initialMatrix = mat4.clone(modelViewMatrix);
    loadTexture(gl, new Uint8Array([200,200,200,255]))
    mat4.translate(modelViewMatrix,modelViewMatrix,[0,-50,0])
    mat4.scale(modelViewMatrix,modelViewMatrix, [10000,5,10000])
    gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false,
modelViewMatrix);
    gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
    mat4.copy(modelViewMatrix, initialMatrix);
    loadTexture(gl, new Uint8Array([50,50,50,255]));
    mat4.translate(modelViewMatrix,modelViewMatrix, [0,-15,0])
    mat4.scale(modelViewMatrix,modelViewMatrix, [50,30,50])
    gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false,
modelViewMatrix);
    gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
    for (var i = -1; i<2; i+=2) {
        mat4.copy(modelViewMatrix, initialMatrix);
        mat4.translate(modelViewMatrix,modelViewMatrix, [0,-40,0])
        mat4.rotate(modelViewMatrix,modelViewMatrix, Math.PI/4, [0,i,0])
        mat4.scale(modelViewMatrix,modelViewMatrix, [350,5,15])
        gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
        gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
        mat4.copy(modelViewMatrix, initialMatrix);
    }
    mat4.rotate(modelViewMatrix,modelViewMatrix,Math.PI*stepperpos/100,[0,-
1,0])
    for (let i = 0; i < 4; i++) {
        const initialMatrix = mat4.clone(modelViewMatrix);
        mat4.translate(modelViewMatrix, modelViewMatrix,
[positions[i][0]*2,positions[i][1]*2,0]);
        mat4.rotate(modelViewMatrix, modelViewMatrix, positions[i][2],
[0,0,1])
        loadTexture(gl, new Uint8Array([50,50,50,255]));
        mat4.scale(modelViewMatrix,modelViewMatrix, [20, positions[i][3], 20])
        gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
        gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
        mat4.copy(modelViewMatrix, initialMatrix);
    }
    loadTexture(gl, new Uint8Array([250,250,250,255]))
    for (let i = -1; i<=1; i+=2) {
        const initialMatrix = mat4.clone(modelViewMatrix);
        mat4.translate(modelViewMatrix, modelViewMatrix,
[positions[4][0]*2+(15*Math.cos(angle)), positions[4][1]*2-20,
i*(15*Math.sin(angle))])
```

```
        mat4.rotate(modelViewMatrix,modelViewMatrix, Math.PI/2, [0,0,1])
        mat4.rotate(modelViewMatrix,modelViewMatrix, i*-angle, [1,0,0])
        mat4.scale(modelViewMatrix,modelViewMatrix, [5, 15, 5])
        gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
        gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
        mat4.copy(modelViewMatrix, initialMatrix);
        mat4.translate(modelViewMatrix, modelViewMatrix,
[positions[4][0]*2+(30*Math.cos(angle)+15*Math.cos(angle-Math.PI/4)),
positions[4][1]*2-20, i*(30*Math.sin(angle)-15*Math.cos(angle+Math.PI/4))])
        mat4.rotate(modelViewMatrix,modelViewMatrix, Math.PI/2, [0,0,1])
        mat4.rotate(modelViewMatrix,modelViewMatrix, i*-(angle-Math.PI/4),
[1,0,0])
        mat4.scale(modelViewMatrix,modelViewMatrix, [5, 15, 5])
        gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
        gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
        mat4.copy(modelViewMatrix, initialMatrix);
    }
    loadTexture(gl, new Uint8Array([0,200,50,255]));
    initialMatrix = mat4.clone(modelViewMatrix);
    mat4.translate(modelViewMatrix,modelViewMatrix,[25,50,0])
    mat4.scale(modelViewMatrix,modelViewMatrix, [2,25,25])
    gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false,
modelViewMatrix);
    gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
    mat4.copy(modelViewMatrix, initialMatrix);
    loadTexture(gl, new Uint8Array([50,50,50,255]));
    mat4.translate(modelViewMatrix,modelViewMatrix,[28,50,0])
    mat4.scale(modelViewMatrix,modelViewMatrix, [2,15,10])
    gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix, false,
modelViewMatrix);
    gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
    mat4.copy(modelViewMatrix, initialMatrix);
    var colors = { "red": [255, 0, 0, 255], "yellow": [255, 200, 0,
255],"green": [0, 150, 150, 255], "blue": [0, 50, 255, 255] };
    for (let i = 0; i<barrels.length; i++) {
        if (barrels[i].attached == "yes") {
            loadTexture(gl, new Uint8Array(colors[barrels[i].color]));
            const initialMatrix = mat4.clone(modelViewMatrix);
            mat4.translate(modelViewMatrix, modelViewMatrix,
[positions[4][0]*2+40, positions[4][1]*2-15, 0])
            mat4.scale(modelViewMatrix,modelViewMatrix,[15,30,15])
            gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
            gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
            mat4.copy(modelViewMatrix, initialMatrix);
        }
```

```
    }
    mat4.copy(modelViewMatrix,realinitialmatrix)
    for (let i = 0; i<barrels.length; i++) {
        if (barrels[i].attached != "yes") {
            console.log(colors[barrels[i].color])
            loadTexture(gl, new
Uint8Array(colors[barrels[i].color]));
            const initialMatrix = mat4.clone(modelViewMatrix);
            mat4.translate(modelViewMatrix,modelViewMatrix,[barrels[i].positio
n[0]*2, -15, barrels[i].position[1]*2])
            mat4.scale(modelViewMatrix,modelViewMatrix,[15,30,15])
            gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
            gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
            mat4.copy(modelViewMatrix, initialMatrix);
        }
    }
}

function drawSceneForPicking(gl: WebGLRenderingContext, programInfo:
ProgramInfo, buffers: Buffers, cameraRotationX: number, cameraRotationY:
number, zoom: number, barrels: Barrel[]) {
    gl.clearColor(1.0, 1.0, 1.0, 1.0);
    gl.clearDepth(1.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
    const fieldOfView = (45 * Math.PI) / 180;
    const canvas = gl.canvas;
    const aspect = canvas.clientWidth / canvas.clientHeight;
    const zNear = 0.1;
    const zFar = 5000.0;
    const projectionMatrix = mat4.create();
    mat4.perspective(projectionMatrix, fieldOfView, aspect, zNear, zFar);
    let modelViewMatrix = mat4.create();
    const xpos = -zoom * Math.sin(cameraRotationX) *
Math.cos(cameraRotationY);
    const ypos = zoom * Math.sin(cameraRotationY);
    const zpos = zoom * Math.cos(cameraRotationX) * Math.cos(cameraRotationY);
    mat4.lookAt(modelViewMatrix, [xpos, ypos, zpos], [0,0,0], [0, 1, 0]);
    gl.useProgram(programInfo.program);
    gl.uniformMatrix4fv(programInfo.uniformLocations.projectionMatrix, false,
projectionMatrix);
    gl.disableVertexAttribArray(programInfo.attribLocations.vertexNormal);
    gl.disableVertexAttribArray(programInfo.attribLocations.textureCoord);
    setPositionAttribute(gl, buffers, programInfo);
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, buffers.indices);
    const vertexCount = 36;
```

```
    const type = gl.UNSIGNED_SHORT;
    const offset = 0;
    for (let i = 0; i<barrels.length; i++) {
        if (barrels[i].attached != "yes") {
            const initialMatrix = mat4.clone(modelViewMatrix);
            loadTexture(gl, new Uint8Array(barrels[i].colorID));
            mat4.translate(modelViewMatrix,modelViewMatrix,[barrels[i].positio
n[0]*2, -15, barrels[i].position[1]*2])
            mat4.scale(modelViewMatrix,modelViewMatrix,[15,30,15])
            gl.uniformMatrix4fv(programInfo.uniformLocations.modelViewMatrix,
false, modelViewMatrix);
            gl.drawElements(gl.TRIANGLES, vertexCount, type, offset);
            mat4.copy(modelViewMatrix, initialMatrix);
        }
    }

}

function setPositionAttribute(gl: WebGLRenderingContext, buffers: Buffers,
programInfo: ProgramInfo) {
    const numComponents = 3;
    const type = gl.FLOAT;
    const normalize = false;
    const stride = 0;
    const offset = 0;
    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position);
    gl.vertexAttribPointer(
        programInfo.attribLocations.vertexPosition,
        numComponents,
        type,
        normalize,
        stride,
        offset,
      );
    gl.enableVertexAttribArray(programInfo.attribLocations.vertexPosition);
}

function setTextureAttribute(gl: WebGLRenderingContext, buffers: Buffers,
programInfo: ProgramInfo) {
    const num = 2;
    const type = gl.FLOAT;
    const normalize = false;
    const stride = 0;
    const offset = 0;
    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.textureCoord);
    gl.vertexAttribPointer(programInfo.attribLocations.textureCoord, num,
type, normalize, stride, offset);
    gl.enableVertexAttribArray(programInfo.attribLocations.textureCoord)
```

```
}

function setNormalAttribute(gl: WebGLRenderingContext, buffers: Buffers,
programInfo: ProgramInfo) {
    const numComponents = 3;
    const type = gl.FLOAT;
    const normalize = false;
    const stride = 0;
    const offset = 0;
    gl.bindBuffer(gl.ARRAY_BUFFER, buffers.normal);
    gl.vertexAttribPointer(programInfo.attribLocations.vertexNormal,
numComponents, type, normalize, stride, offset,);
    gl.enableVertexAttribArray(programInfo.attribLocations.vertexNormal);
}

function loadTexture(gl: WebGLRenderingContext, color: ArrayBuffer) {
    const texture = gl.createTexture();
    gl.bindTexture(gl.TEXTURE_2D, texture);
    const level = 0;
    const internalFormat = gl.RGBA;
    const width = 1;
    const height = 1;
    const border = 0;
    const srcFormat = gl.RGBA;
    const srcType = gl.UNSIGNED_BYTE;
    const pixel = new Uint8Array(color);
    gl.texImage2D(gl.TEXTURE_2D, level, internalFormat, width, height, border,
srcFormat, srcType, pixel);
    return texture;
}


export { drawScene, drawSceneForPicking };
```

7.1.1.2 main.ts

```
import { Buffers, initBuffers } from "./cube-buffer.js";
import { drawScene, drawSceneForPicking } from "./draw-scene.js";

var mousedown = false;
var mousePos = {x:0,y:0}
var prevmouse = {x:0,y:0}
var positions = [[0, 29.75, 0.0, 59.5+4], [6.283683576271408,
92.71077115505295, 2.9545969675064323, 67.6+4], [43.78368357627141,
112.96077115505295, 1.1772622130201693, 67.6+4], [87.5, 100.0,
1.5707963267948966, 25],[100,100]]
var angle = Math.PI/4;
var stepperpos = 100
var barrels: Barrel[] = [];
```

```
var websocket = new WebSocket("ws://192.168.137.81:8765")

function updateInfo() {
    const x: string = (document.getElementById("xpos") as
HTMLInputElement).value;
    const y: string = (document.getElementById("ypos") as
HTMLInputElement).value;
    const step: string = (document.getElementById("step") as
HTMLInputElement).value;
    websocket.send(`${x} ${y} ${step}`);
}

function run() {
    websocket.send("code: "+(document.getElementById("code") as
HTMLInputElement).value)
}

export type Barrel = {
    position: number[],
    colorID: number[],
    attached: string,
    color: string
}

function takePhoto() {
    websocket.send("photo")
}

function scan() {
    websocket.send("scan")
}

function moveClaw(checkbox: HTMLInputElement) {
    if (checkbox.checked) {
        angle = Math.PI/2
        websocket.send("claw 45")
    } else {
        websocket.send("claw 0")
        angle = Math.PI/4
    }
}

websocket.onmessage = (event) => {
    if (event.data == "image") {
        const img = document.getElementById('camera') as HTMLImageElement;
        img.src = `static/image.jpg?${Math.ceil(Math.random() *
1000)}`;
    } else if (event.data == "error") {
```

```
        (document.getElementById("error") as HTMLElement).textContent =
"Cannot reach X: " +(document.getElementById("xpos") as
HTMLInputElement).value+" Y: "+(document.getElementById("ypos") as
HTMLInputElement).value;
    } else if (event.data == "barrelerror") {
        (document.getElementById("error") as HTMLElement).textContent =
"Cannot reach barrel";
        for (let i = 0; i<barrels.length; i++) {
            if (barrels[i].attached == "next") {
                barrels[i].attached = "no"
            }
        }
    } else if (event.data.startsWith("barrelnext")) {
        barrels[parseInt(event.data.substr(event.data.indexOf(" ")
+1))].attached = "next";
    } else if (event.data.startsWith("output")) {
        const actualout = event.data.substr(event.data.indexOf(" ") + 1)
        if (actualout == "clear") {
            (document.getElementById("output") as HTMLElement).innerHTML=""
        } else {
            (document.getElementById("output") as HTMLElement).innerHTML =
(document.getElementById("output") as HTMLElement).innerHTML +
"<br/>"+actualout
        }
    } else if (event.data.startsWith("barrel ")) {
        var info = event.data.split(" ")
        barrels.push({position: [parseFloat(info[1]),parseFloat(info[2])],
colorID:[255,barrels.length*10,0,255], attached:"no", color: info[3]})
    } else if (event.data.startsWith("stepperpos")) {
        stepperpos=parseInt(event.data.split(" ")[1])
        console.log(stepperpos)
    } else if (event.data.startsWith("claw ")) {
        angle=parseFloat(event.data.split(" ")[1])
    } else if (event.data == "attached") {
        for (var i = 0; i<barrels.length; i++) {
            if (barrels[i].attached == "next") {
                barrels[i].attached = "yes"
            }
        }
    } else if (event.data.startsWith("dropped")) {
        var info = event.data.split(" ");
        for (var i = 0; i < barrels.length; i++) {
            if (barrels[i].attached == "yes") {
                barrels[i].attached = "no";
                barrels[i].position = [parseFloat(info[1]),
parseFloat(info[2])]
            }
        }
```

```
    }
    else if (event.data == "clearscan") {
        var newbarrels = []
        for (var i = 0; i<barrels.length; i++) {
            if (barrels[i].attached == "yes") {
                newbarrels.push(barrels[i])
            }
        }
        barrels = newbarrels
    } else {
        (document.getElementById("error") as HTMLElement).textContent = "";
        var obj  = JSON.parse(event.data);
        positions = obj
    }
};

(window as any).updateInfo = updateInfo;
(window as any).moveClaw = moveClaw;
(window as any).takePhoto = takePhoto;
(window as any).scan = scan;
(window as any).run = run;

main();

export type ProgramInfo = {
    program: WebGLProgram;
    attribLocations: {
        vertexPosition: number;
        vertexNormal: number;
        textureCoord: number;
    };
    uniformLocations: {
        projectionMatrix: WebGLUniformLocation;
        modelViewMatrix: WebGLUniformLocation;
        normalMatrix: WebGLUniformLocation;
        uSampler: WebGLUniformLocation;
    };
}

function main() {
    const canvas = document.querySelector("#glcanvas")
    const gl = (canvas as HTMLCanvasElement).getContext("webgl") as
WebGLRenderingContext;
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.clear(gl.COLOR_BUFFER_BIT);
    const vsSource = `
        attribute vec4 aVertexPosition;
        attribute vec3 aVertexNormal;
```

```
        attribute vec2 aTextureCoord;
        uniform mat4 uNormalMatrix;
        uniform mat4 uModelViewMatrix;
        uniform mat4 uProjectionMatrix;
        varying highp vec2 vTextureCoord;
        varying highp vec3 vLighting;
        void main(void) {
            gl_Position = uProjectionMatrix * uModelViewMatrix *
aVertexPosition;
            vTextureCoord = aTextureCoord;
            highp vec3 ambientLight = vec3(0.3, 0.3, 0.3);
            highp vec3 directionalLightColor = vec3(1, 1, 1);
            highp vec3 directionalVector = normalize(vec3(0.85, 0.8, 0.75));
            highp vec4 transformedNormal = uNormalMatrix * vec4(aVertexNormal,
1.0);
            highp float directional = max(dot(transformedNormal.xyz,
directionalVector), 0.0);
            vLighting = ambientLight + (directionalLightColor * directional);
        }
    `;
    const fsSource = `
        varying highp vec2 vTextureCoord;
        varying highp vec3 vLighting;
        uniform sampler2D uSampler;
        void main(void) {
        highp vec4 texelColor = texture2D(uSampler, vTextureCoord);
        gl_FragColor = vec4(texelColor.rgb * vLighting, texelColor.a);
        }
    `;
    const shaderProgram = initShaderProgram(gl, vsSource, fsSource);
    const programInfo: ProgramInfo = {
        program: shaderProgram,
        attribLocations: {
          vertexPosition: gl.getAttribLocation(shaderProgram,
"aVertexPosition"),
          vertexNormal: gl.getAttribLocation(shaderProgram, "aVertexNormal"),
          textureCoord: gl.getAttribLocation(shaderProgram, "aTextureCoord"),
        },
        uniformLocations: {
          projectionMatrix: gl.getUniformLocation(shaderProgram,
"uProjectionMatrix") as WebGLUniformLocation,
          modelViewMatrix: gl.getUniformLocation(shaderProgram,
"uModelViewMatrix") as WebGLUniformLocation,
          normalMatrix: gl.getUniformLocation(shaderProgram, "uNormalMatrix")
as WebGLUniformLocation,
          uSampler: gl.getUniformLocation(shaderProgram, "uSampler") as
WebGLUniformLocation,
        },
```

```
      };
    const buffers = initBuffers(gl) as Buffers;
    gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
    function render() {
        drawScene(gl, programInfo, buffers, mousePos.x, mousePos.y, positions,
1800, angle, barrels, stepperpos);
        requestAnimationFrame(render);
    }
    requestAnimationFrame(render);
    (canvas as HTMLCanvasElement).addEventListener('mousemove', function(e) {
        getMousePosition(e);
    });
    addEventListener("click", function(e: MouseEvent) {
        const target = e.target as HTMLButtonElement
        const rect = target.getBoundingClientRect();
        const x = e.clientX - rect.left;
        const y = e.clientY - rect.top;
        console.log(x,y)
        drawSceneForPicking(gl, programInfo, buffers, mousePos.x, mousePos.y,
1800, barrels);
        var pixels = new Uint8Array(4);
        gl.readPixels(x, rect.height - y, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE,
pixels);
        const scale = 255/Math.max(...pixels.subarray(0,3))
        for (let i = 0; i<pixels.length; i++) {
            pixels[i]=pixels[i]*scale
        }
        console.log(pixels)
        if (pixels[0] == 255 && pixels[2] == 0) {
            websocket.send("barrel
"+barrels[Math.round(pixels[1]/10)].position[0]+"
"+barrels[Math.round(pixels[1]/10)].position[1])
            barrels[Math.round(pixels[1]/10)].attached="next"
        }
    });
}


function initShaderProgram(gl: WebGLRenderingContext, vsSource: string,
fsSource: string) {
    const vertexShader = loadShader(gl, gl.VERTEX_SHADER, vsSource) as
WebGLShader;
    const fragmentShader = loadShader(gl, gl.FRAGMENT_SHADER, fsSource) as
WebGLShader;
    const shaderProgram = gl.createProgram() as WebGLProgram;
    gl.attachShader(shaderProgram, vertexShader);
    gl.attachShader(shaderProgram, fragmentShader);
    gl.linkProgram(shaderProgram);
    return shaderProgram;
```

```
}

function loadShader(gl: WebGLRenderingContext, type: GLenum, source: string) {
    const shader = gl.createShader(type) as WebGLShader;
    gl.shaderSource(shader, source);
    gl.compileShader(shader);
    return shader;
}

addEventListener("mousedown", (event) => {
    mousedown = true;
    const rect = (event.target as HTMLElement).getBoundingClientRect();
    prevmouse.x = ((event.clientX - rect.left) / rect.width * 2 - 1) * 4;
    prevmouse.y = ((event.clientY - rect.top) / rect.height * 2 - 1) * 2;
});

addEventListener("mouseup", (_) => {
    mousedown = false;
});

 function getMousePosition(event: MouseEvent) {
    const target = event.currentTarget as HTMLButtonElement
    if (mousedown) {
        const rect = target.getBoundingClientRect();
        const x = ((event.clientX - rect.left) / rect.width * 2 - 1)*4;
        const y = ((event.clientY - rect.top) / rect.height * 2 - 1)*2;
        mousePos = {x: mousePos.x+x-prevmouse.x, y: mousePos.y+y-prevmouse.y}
        prevmouse = {x: x,y: y}
    }
}
```

7.1.1.3 cube-buffer.ts

```
export type Buffers = {
  position: WebGLBuffer;
  normal: WebGLBuffer;
  textureCoord: WebGLBuffer;
  indices: WebGLBuffer;
}

function initBuffers(gl: WebGLRenderingContext) {
    const positionBuffer = initPositionBuffer(gl);
    const textureCoordBuffer = initTextureBuffer(gl);
    const indexBuffer = initIndexBuffer(gl);
    const normalBuffer = initNormalBuffer(gl);
    return { position: positionBuffer, normal: normalBuffer, textureCoord:
textureCoordBuffer, indices: indexBuffer };
}
```

```
function initPositionBuffer(gl: WebGLRenderingContext) {
    const positionBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
    const positions = [
        -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0, 1.0,
        -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0,
        -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0,
        -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0,
        1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, 1.0, 1.0, 1.0, -1.0, 1.0,
        -1.0, -1.0, -1.0, -1.0, -1.0, 1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(positions),
gl.STATIC_DRAW);
    return positionBuffer;
}

function initTextureBuffer(gl: WebGLRenderingContext) {
    const textureCoordBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, textureCoordBuffer);
    const textureCoordinates = [
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
      0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0,
    ];

    gl.bufferData(
      gl.ARRAY_BUFFER,
      new Float32Array(textureCoordinates),
      gl.STATIC_DRAW,
    );

    return textureCoordBuffer;
  }

function initIndexBuffer(gl: WebGLRenderingContext) {
    const indexBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);
    const indices = [
      0, 1, 2, 0, 2, 3,
      4, 5, 6, 4, 6, 7,
      8, 9, 10, 8, 10, 11,
      12, 13, 14, 12, 14, 15,
      16, 17, 18, 16, 18, 19,
      20, 21, 22, 20, 22, 23,
    ];
```

```
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices),
gl.STATIC_DRAW,);
    return indexBuffer;
}

function initNormalBuffer(gl: WebGLRenderingContext) {
    const normalBuffer = gl.createBuffer();
    gl.bindBuffer(gl.ARRAY_BUFFER, normalBuffer);
    const vertexNormals = [
        0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0,
        0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0,
        0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0,
        0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0,
        1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0,
        -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0, -1.0, 0.0, 0.0,
    ];
    gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexNormals),
gl.STATIC_DRAW,);
    return normalBuffer;
  }


export { initBuffers };
```

7.1.2 Other files

7.1.2.1 SampleCode.txt

```
barrels = await scan()
for i in range(len(barrels)):
    await output("starting barrel " +str(i)+" ("+barrels[i].color+")")
    barrels = await pickup(barrels, i)
    await rotate((i+1)*50)
    await move(180, 60)
    await move(210, 50)
    barrels = await drop(barrels)
    await move(160, 30)
    await move(160, 200)
    await output(str(i)+" is complete")
```

7.1.2.2 tsconfig.json

```
{
    "compilerOptions": {
      "target": "ES6",
      "module": "ESNext",
      "moduleResolution": "Node",
      "strict": true,
      "esModuleInterop": true,
      "skipLibCheck": true,
      "outDir": "./static",
      "rootDir": "./src"
    },
    "include": ["src/**/*"],
    "exclude": ["node_modules"]
}
```

7.1.2.3 Index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>WebGL Demo</title>
    <style>
      #controls {
        position: absolute;
        left: 20px;
        top: 20px;
      }
      #controls label,
```

```
      #controls input,
      #controls button {
        display: block;
        margin-bottom: 5px;
      }
      #error {
        color: red;
      }
      #sidebar {
        position: absolute;
        left: 1100px;
        top: 10px;
      }
      #output {
        position:absolute;
        top: 0px;
        width: 180px;
        left: 500px;
      }
      #run {
        position: absolute;
        left: 10px;
        top:200px;
      }
      #docs {
        position:absolute;
        left: 50px;
        top: 200px;
      }
      #camera {
        position: absolute;
        left: 0px;
        top:240px
      }
    </style>
    <script
      src="https://cdnjs.cloudflare.com/ajax/libs/gl-matrix/2.8.1/gl-matrix-
min.js"
      integrity="sha512-
zhHQR0/H5SEBL3Wn6yYSaTTZej12z0hVZKOv3TwCUXT1z5qeqGcXJLLrbERYRScEDDpYIJhPC1fk31
gqR783iQ=="
      crossorigin="anonymous"
      defer></script>
      <script src="https://code.jquery.com/jquery-3.2.1.min.js"
integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
crossorigin="anonymous"></script>
    <script src="/static/main.js" type="module"></script>
  </head>
```

```
  <body>
    <canvas id="glcanvas" width="1080" height="720"></canvas>
    <div id="sidebar">
      <textarea id="code" rows="12" cols="60"></textarea>
      <div id = "output"></div>
      <button id="run" onclick="run()">Run</button>
      <a href="/docs/index.html" id="docs" target="_blank">View
documentation</a>
      <img id="camera" src="" width="640" height="480">
    </div>
    <div id="controls">
      <label for="xpos">X Position:</label>
      <input type="text" id="xpos" value="200">

      <label for="ypos">Y Position:</label>
      <input type="text" id="ypos" value="150">

      <label for="step">Stepper position:</label>
      <input type="text" id="step" value="100">

      <button id="updateButton" onclick="updateInfo()">Update</button>

      <label for="claw">Open Claw:</label>
      <input type="checkbox" id="claw" name="Open claw"
onclick="moveClaw(this)"/>

      <button id="photoButton" onclick="takePhoto()">Take Photo</button>
      <button id="scan" onclick="scan()">Scan</button>

      <div id = "error"></div>
    </div>
  </body>
</html>
```

7.2 Server code

7.2.1 Python files

7.2.1.1 Server.py

```
import asyncio
import websockets
from aiohttp import web
from arm_info import Arm, Beam
import math
import json
from camera import Camera
from time import sleep
from barrel import Barrel
from parse import parse
from movement import scan, pickup, drop

async def echo(websocket: websockets.WebSocketServerProtocol) -> None:
    arm = Arm(Beam(98.5,135,0),
Beam(101.5,135,0),Beam(81.5,135,0),Beam(45.5,180,0,math.radians(90)))
    camera = Camera()
    barrels: list[Barrel] = []
    async for message in websocket:
        if message[:len("code: ")] == "code: ":
            barrels = await parse(arm, camera, websocket, barrels,
message[len("code: "):])
            continue
        print(message)
        message = message.split()
        if message[0] == "claw":
            if message[1] == "45":
                barrels = await drop(arm, websocket, barrels)
            arm.move_claw(float(message[1]))
        elif message[0] == "photo":
            print("photo requested")
            camera.takePhoto()
            await websocket.send("image")
        elif message[0] == "scan":
            await websocket.send("clearscan")
            barrels = await scan(arm, camera, websocket, barrels)
        elif message[0] == "barrel":
            for i in range(len(barrels)):
                if barrels[i].x == int(message[1]) and barrels[i].y ==
int(message[2]):
                    print("going to barrel")
                    barrels = await pickup(arm, camera, websocket, barrels, i)
        else:
```

```
            positions = arm.setPosition(float(message[0]),float(message[1]))
            sleep(2)
            arm.setStepper(int(message[2]))
            if positions != []: await websocket.send(json.dumps(positions))
            else: await websocket.send("error")
            await websocket.send("stepperpos "+str(200-arm.stepperPos))


async def handle_html(_) -> web.FileResponse:
        return web.FileResponse('./index.html')

async def start_servers():
    webocket_server = websockets.serve(echo, "192.168.137.81", 8765)
    app = web.Application()
    app.router.add_get('/', handle_html)
    app.router.add_static('/static/', path='./static', name='static')
    app.router.add_static('/docs', path='./mkdocs/site', name='docs',
show_index=True)
    runner = web.AppRunner(app)
    await runner.setup()
    site = web.TCPSite(runner, "192.168.137.81", 8080)
    await asyncio.gather(webocket_server, site.start())

asyncio.get_event_loop().run_until_complete(start_servers())
asyncio.get_event_loop().run_forever()
```

7.2.1.2 Parse.py

```
import movement
from arm_info import Arm
from camera import Camera
from barrel import Barrel
import websockets

async def parse(arm: Arm, camera: Camera, websocket:
websockets.WebSocketServerProtocol, barrels: list[Barrel], code: str) ->
list[Barrel]:
    print(code)
    scan = lambda *args: movement.scan(arm, camera, websocket, *args)
    pickup = lambda *args: movement.pickup(arm, camera, websocket, *args)
    drop = lambda *args: movement.drop(arm, websocket, *args)
    rotate = lambda *args: movement.rotate(arm, websocket, *args)
    move = lambda *args: movement.move(arm, websocket, *args)
    async def output(x: str) -> None:
        print("outputting",x)
        await websocket.send("output "+x)
    await output("clear")
    code = (
        "try:\n" +
```

```
        ''.join(f'    {line}\n' for line in code.split('\n')) +
        "except Exception as e:\n"
        "    await output(f'Error: {str(e)}')"
    )
    exec(
        f'async def __ex(): ' +
        ''.join(f'\n {l}' for l in code.split('\n'))
        , locals()
    )
    await locals()['__ex']()
    await output("done :)")
    print("done")
    return barrels
```

7.2.1.3 Movement.py

```
from time import sleep
import json
from barrel import Barrel
import math
from arm_info import Arm
from camera import Camera
import websockets

async def scan(arm: Arm, camera: Camera, websocket:
websockets.WebSocketServerProtocol, barrels: list[Barrel] = []) ->
list[Barrel]:
        barrels = [i for i in barrels if i.gripped]
        position = arm.setPosition(100.0,250.0)
        await websocket.send(json.dumps(position))
        sleep(2)
        end = arm.stepperPos+200
        while arm.stepperPos < end:
            arm.setStepper(arm.stepperPos)
            await websocket.send("stepperpos "+str(200-arm.stepperPos))
            sleep(2)
            distance,color =camera.distance()
            print(distance,color)
            while 90 < distance < 300 and (5 < arm.stepperPos%200 < 195):
                move = camera.getCentral()
                print(distance,move)
                if abs(move) <= 100:
                    barrels.append(Barrel(arm.stepperPos, distance, color))
                    [print(i.x,i.y, i.distance,i.angle) for i in barrels]
                    await websocket.send("barrel "+barrels[-1].getData())
                    arm.stepperPos+=25
                    break
                if arm.stepperPos>end:
```

```
                    break
                arm.stepperPos = arm.stepperPos-int(move/100)
                arm.setStepper(arm.stepperPos)
                await websocket.send("stepperpos "+str(200-arm.stepperPos))
                sleep(2)
                distance, color = camera.distance()
                print(distance)
            arm.stepperPos+=10
            sleep(2)
        return barrels

async def pickup(arm: Arm, camera, websocket:
websockets.WebSocketServerProtocol, barrels: list[Barrel], i: int) ->
list[Barrel]:
    await websocket.send("barrelnext "+str(i))
    if not (170 <= barrels[i].distance <= 210):
         await websocket.send("barrelerror")
         return barrels
    success = False
    while not success:
        position = arm.setPosition(100.0,250.0)
        await websocket.send(json.dumps(position))
        sleep(2)
        arm.move_claw(45)
        await websocket.send("claw "+str(math.pi/2))
        sleep(2)
        arm.setStepper(barrels[i].angle)
        arm.stepperPos = barrels[i].angle
        await websocket.send("stepperpos "+str(200-arm.stepperPos))
        sleep(2)
        position = arm.setPosition(160.0, 30.0)
        await websocket.send(json.dumps(position))
        sleep(2)
        position = arm.setPosition(barrels[i].distance, 50.0)
        print("important stuff", position, barrels[i].distance)
        await websocket.send(json.dumps(position))
        sleep(2)
        arm.move_claw(0)
        await websocket.send("claw "+str(math.pi/4))
        sleep(2)
        position = arm.setPosition(150.0,200.0)
        await websocket.send(json.dumps(position))
        print("has picked up?", camera.getCentral(), camera.distance()[0],
barrels[i].distance)
        if not (camera.getCentral()<100 and abs(camera.distance()[0]-
barrels[i].distance)<50):
            success= True
            barrels[i].gripped = True
```

```
            await websocket.send("attached")
            return barrels
        print("retrying")


async def drop(arm: Arm, websocket: websockets.WebSocketServerProtocol,
barrels: list[Barrel]) -> list[Barrel]:
    for i in range(len(barrels)):
        if barrels[i].gripped==True:
            barrels[i] = Barrel(arm.stepperPos, arm.beam3.endx,
barrels[i].color)
            print(arm.beam3.endy)
            print(barrels[i].getData())
            await websocket.send("dropped "+barrels[i].getData())
    arm.move_claw(45)
    return barrels

async def move(arm: Arm, websocket: websockets.WebSocketServerProtocol, x, y)
-> None:
    positions = arm.setPosition(x,y)
    await websocket.send(json.dumps(positions))
    sleep(2)

async def rotate(arm: Arm, websocket: websockets.WebSocketServerProtocol,
stepperpos: int) -> None:
    arm.setStepper(stepperpos)
    await websocket.send("stepperpos "+str(200-arm.stepperPos))
    sleep(2)
```

7.2.1.4 Camera.py

```
from picamera2 import Picamera2
import cv2
import numpy as np


class Camera:
    def __init__(self) -> None:
        self.camera = Picamera2()
        self.camera.configure(self.camera.create_still_configuration())
        self.camera.start()

    def takePhoto(self) -> None:
        self.camera.capture_file("static/image.jpg")

    def returnPhoto(self) -> np.ndarray:
        return self.camera.capture_array()

    def getContours(self) -> tuple[np.ndarray | int, str]:
```

```
        image = self.camera.capture_array()
        hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
        maxContourArea = 0
        contour = (-1,"red")
        colors = {
            "red": [(np.array([0, 150, 80]), np.array([10, 255, 255])),
                    (np.array([170, 150, 80]), np.array([180, 255, 255]))],
            "yellow": [(np.array([10, 150, 80]), np.array([55, 255, 255]))],
            "green": [(np.array([70, 150, 80]), np.array([100, 255, 255]))],
            "blue": [(np.array([100, 150, 80]), np.array([130, 255, 255]))]
        }

        for color in colors:
            mask = cv2.inRange(hsv_image, colors[color][0][0],
colors[color][0][1])
            if len(colors[color]) == 2:
                mask = cv2.bitwise_or(mask, cv2.inRange(hsv_image,
colors[color][1][0], colors[color][1][1]))
            contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
            if len(contours) != 0:
                contours = sorted(contours, key=cv2.contourArea, reverse=True)
                if cv2.contourArea(contours[0]) > maxContourArea:
                    maxContourArea = cv2.contourArea(contours[0])
                    contour = (contours[0],color)
        return contour

    def distance(self) -> tuple[float, str]:
        contours,color = self.getContours()
        if type(contours) != int:
            w = cv2.boundingRect(contours)[2]
            distance = 46600/w
            return distance-40, color
        else:
            return 0, "red"

    def getCentral(self) -> int:
        contours, _ = self.getContours()
        if type(contours) != int:
            rect = cv2.boundingRect(contours)
            difference = (3280/2)-(rect[0]+(rect[2]/2))
            return int(difference)
        return 20000
```

7.2.1.5 Barrel.py

```
import math

class Barrel:
    def __init__(self, angle: int, distance: float, color: str) -> None:
        self.x: int = round(distance*math.cos(math.radians(angle*1.8)))
        self.y: int = round(-distance*math.sin(math.radians(angle*1.8)))
        self.angle = angle
        self.distance = distance
        self.gripped: bool = False
        self.color: str = color

    def getData(self) -> str:
        return str(self.x)+" "+str(self.y)+" "+self.color
```

7.2.1.6 Arm_info.py

```
import math
import serial
from time import sleep

def law_of_cosines(a: float, b: float, c: float) -> float:
    return math.acos((a * a + b * b - c * c) / (2 * a * b))

def distance(x: float, y: float) -> float:
    return math.sqrt(x * x + y * y)

class Beam:
    def __init__(self, length: float, tmax: int, tmin: int, absolute = 0.0):
        self.length = length
        self.absolute = absolute
        self.tmin = tmin
        self.tmax = tmax
        self.t: float = 0
        self.endx: float = 0
        self.endy: float = 0
        self.centerx: float = 0
        self.centery: float = length/2
        self.angle = 0
    def possible(self) -> bool:
        return self.tmin <= self.angle <= self.tmax


class Arm:
    def __init__(self, beam0: Beam, beam1: Beam, beam2: Beam, beam3: Beam,
clawOpen = 0) -> None:
        self.beam0 = beam0
```

```
        self.beam1 = beam1
        self.beam2 = beam2
        self.beam3 = beam3
        self.clawOpen = clawOpen
        self.arduino = serial.Serial('/dev/ttyUSB0', baudrate=115200)
        self.stepperPos = 100

    def setPosition(self, x: float, y: float) -> list[list[float]]:
        try:
            y = y-self.beam0.length
            x = x-self.beam3.length
            dist: float = distance(x, y)
            D1 = math.atan2(y, x)
            D2 = law_of_cosines(dist, self.beam1.length, self.beam2.length)
            self.beam1.t = D1 + D2
            self.beam1.absolute = self.beam1.t+math.radians(90)
            self.beam2.t = law_of_cosines(self.beam1.length,
self.beam2.length, dist)
            self.beam2.absolute = (self.beam1.absolute+self.beam2.t)-
math.radians(180)
            self.beam1.endx = self.beam1.length * math.cos(self.beam1.t)
            self.beam1.endy = self.beam1.length *
math.sin(self.beam1.t)+self.beam0.length
            self.beam2.endx = self.beam1.endx - self.beam2.length *
math.cos(self.beam1.t + self.beam2.t)
            self.beam2.endy = self.beam1.endy - self.beam2.length *
math.sin(self.beam1.t + self.beam2.t)
            self.beam3.endx = self.beam2.endx+self.beam3.length
            self.beam1.centerx = (self.beam1.endx)/2
            self.beam1.centery = (self.beam1.endy+self.beam0.length)/2
            self.beam2.centerx = (self.beam1.endx+self.beam2.endx)/2
            self.beam2.centery = (self.beam1.endy+self.beam2.endy)/2
            self.beam3.centerx = (self.beam2.endx*2+self.beam3.length)/2
            self.beam3.centery = self.beam2.endy
            self.beam1.angle = 135-int(math.degrees(self.beam1.t))
            self.beam2.angle = 180-int(math.degrees(self.beam2.t)-45)
            self.beam3.angle = int(math.degrees(self.beam1.t+self.beam2.t)-90)
            if not (self.beam1.possible() and self.beam2.possible() and
self.beam3.possible()): return []
            self.sendToArduino()
            return [[self.beam0.centerx, self.beam0.centery,
self.beam0.absolute, self.beam0.length],
                    [self.beam1.centerx, self.beam1.centery,
self.beam1.absolute, self.beam1.length],
                    [self.beam2.centerx, self.beam2.centery,
self.beam2.absolute, self.beam2.length],
                    [self.beam3.centerx, self.beam3.centery,
self.beam3.absolute, self.beam3.length],
```

```
                [self.beam2.endx+self.beam3.length, self.beam2.endy]]
        except: return []
    def setStepper(self, pos: int) -> None:
        self.stepperPos =pos
        print(pos%200)
        self.arduino.write(bytes("5"+str(self.stepperPos%200).zfill(3)+"\n",'u
tf-8'))

    def plotInfo(self) -> list[list[float]]:
        return [[0,0, self.beam1.endx, self.beam2.endx,
self.beam2.endx+self.beam3.length], [0, self.beam0.length, self.beam1.endy,
self.beam2.endy, self.beam2.endy]]

    def sendToArduino(self) -> None:
        print("3"+ str(self.beam3.angle).zfill(3) + "1" +
str(self.beam1.angle).zfill(3) + "2" + str(self.beam2.angle).zfill(3))
        self.arduino.write(bytes("3" + str(self.beam3.angle).zfill(3) + "2" +
str(self.beam2.angle).zfill(3) + "1" + str(self.beam1.angle).zfill(3) + "\n",
'utf-8'))

    def move_claw(self, claw_pos):
        self.clawOpen = claw_pos
        self.arduino.write(bytes("4" + str(claw_pos).zfill(3) + "\n", 'utf-
8'))
        sleep(2)
```

7.2.2 Arduino code

7.2.2.1 arduino_code.cpp

```
#include <Servo.h>
const int servoCount = 4;
const int pwmPins[4] = {3,5,6,9};
Servo servo[servoCount];
#define STEP_PIN 11
#define DIR_PIN 10
String data;
int pos = 100;

void setup()
{
  Serial.begin(115200);
  for (int i = 0; i<servoCount; i++) {
    servo[i].attach(pwmPins[i]);
  }
  pinMode(STEP_PIN, OUTPUT);
  pinMode(DIR_PIN, OUTPUT);
}
```

```
void  loop() {
  while (!Serial.available());
  data = Serial.readString();
  while (data.length() >= 3) {
    Serial.println(data);
    if (data.substring(0,1).toInt() == 5) {
      step(data.substring(1,4).toInt());
    } else {
      servo[data.substring(0,1).toInt()-
1].write(data.substring(1,4).toInt());
      delay(250);
    }
    data = data.substring(4);
  }
}

void step(int go) {
  if (pos-go > 0) {
    digitalWrite(DIR_PIN, HIGH);
  } else {
    digitalWrite(DIR_PIN, LOW);
  }
  for (int i = 0; i < abs(pos-go); i++) {
    digitalWrite(STEP_PIN, HIGH);
    delayMicroseconds(5000);
    digitalWrite(STEP_PIN, LOW);
    delayMicroseconds(5000);
  }
  pos = go;
}
```

7.3 Documentation files

7.3.1 Markdown

7.3.1.1 barrel.md

```
# Barrel

Barrels should not be manually created, but retrieving their properties is
often useful.

## Methods

    self.angle: int
    self.distance: float
```

The angle (in steps, so 200 per full rotation) of the stepper when the barrel is directly in the center of its vision, and the distance the barrel is away from the pivot.

    self.x: int
    self.y: int

Polar to cartesian conversions of angle and distance

    self.gripped: bool = False

Whether the barrel is currently held by the claw, in which case the position information is incorrect.

    self.color: str

The colour of the barrel (red, yellow or blue)

Return data

    getData(self) -> str

returns the x, y and colour in a space-delimited string (used for websocket transmission)

7.3.1.2 camera.md

Camera

An instance of the camera called camera has already been created. Do not attempt to re-initialise the class.

Take photo

    camera.takePhoto(self) -> None

Takes a picture and saves it to static/image.jpg. This will not update the webserver until you send a command via websocket notifying the client of the update.

    camera.takePhoto()
    await websocket.send("image")

This cannot be done repeatedly with very little delay as it causes read/write conflicts on the file.

Distance

    camera.distance(self) -> tuple(float, str)

This returns the distance of the nearest red, yellow or blue object and its color (this is likely a barrel if the value is reasonable, and empty space if not)

Centre

```
camera.getCentral(self) -> int
```

Returns the number of pixels the center of the detected object is from the center (less than 100 generally means it is centered)

7.3.1.3 index.md

Home

The arm runs on Python, and most of your code will be parsed just as any other Python code. The only difference is a few pre-written methods and access to the "Barrel" and "Camera" classes. The high-level movement functions provide most of the functionality that the arm offers, although there is also the option to manually control the arm via the arm class (NOTE: calling these directly will not update the web render, and there is no documentation).

Visit the [GitHub repo](https://github.com/ice-cube-1/robotic-arm) for the full source code, or see how to use some of the arm's [functions](pre-done_functions.md).

7.3.1.4 pre-done_functions.md

Pre-done functions

Some extra points to note about using these:

* Any function defined must be asynchronous as the code is all run within an async function. Likewise, any of the below functions must be called with await.

```
async def x():
    await output("hello")
await x()
```

* These functions have extra parameters that are automatically passed in from your code. These are the arm, the websocket and the camera (scan and pickup only).

Scan

This scans a full 360 degrees ands adds any barrels it detects to the render.

```
async def scan(barrels: list[Barrel] = []) -> list[Barrel]
```

This works as follows:

1. The arm moves up, giving it an unobstructed view of the ground
2. It continuously rotates until the calculated distance to the nearest barrel is within 90 < distance < 300 (it believes a barrel to be somewhere in frame).
3. It attempts to center on the barrel
4. The barrel's colour and position is then added to the list of barrels, and sent back to the website via websocket
5. This repeats until it has reached the start point, when it returns a list of all barrels detected

Any barrels passed in will be removed unless they are currently gripped.

Pickup
From the list of barrels passed in, it picks up the one at index i.

```
    async def pickup(barrels: list[Barrel], i: int) -> list[Barrel]
```

This moves the arm out of the way before rotating, then recenters on the barrel (in case accidental movement has offset the position slightly) and reaches down to pick it up. It then moves back up, and will repeat the whole proess if the barrel is still there.

Drop
Drops the current barrel, presuming it falls directly down and lands upright.

```
    async def drop(barrels: list[Barrel]) -> list[Barrel]
```

This is just a wrapper for the open claw function, but deals with sending data via websocket as well. This requires the arm to currently be very close to the ground to work successfully, as the arm does not move down automatically.

Move
Moves the arm to the requested x (horizontal distance from pivot, mm) and y (vertical distance from pivot), without any rotation.

```
    async def move(x: float, y: float) -> None
```

Rotate
Rotates the entire arm, taking into account that it cannot continuously rotate in one direction. A full 360 degree rotation is divided into 200 steps.

```
    async def rotate(steps: int) -> None
```

Output

Print statements do not work in this UI, so this is the asynchronous equivalent. These show up to the right of the code input, acting as a basic console.

```
    async def output(x: str) -> None
```

7.3.2 Config

7.3.2.1 mkdocs.yaml

```yaml
site_name: Coding the arm
theme:
  name: mkdocs
  color_mode: dark
extra_javascript:
 - add_index.js
nav:
 - index.md
 - pre-done_functions.md
 - barrel.md
 - camera.md
```

7.3.2.2 add_index.js

```javascript
var links = document.getElementsByTagName("a");
for (var i = 0; i < links.length; i++) {
    if (!links[i].href.startsWith("https")) {
        links[i].href = links[i].href+"index.html"
    }
}
```