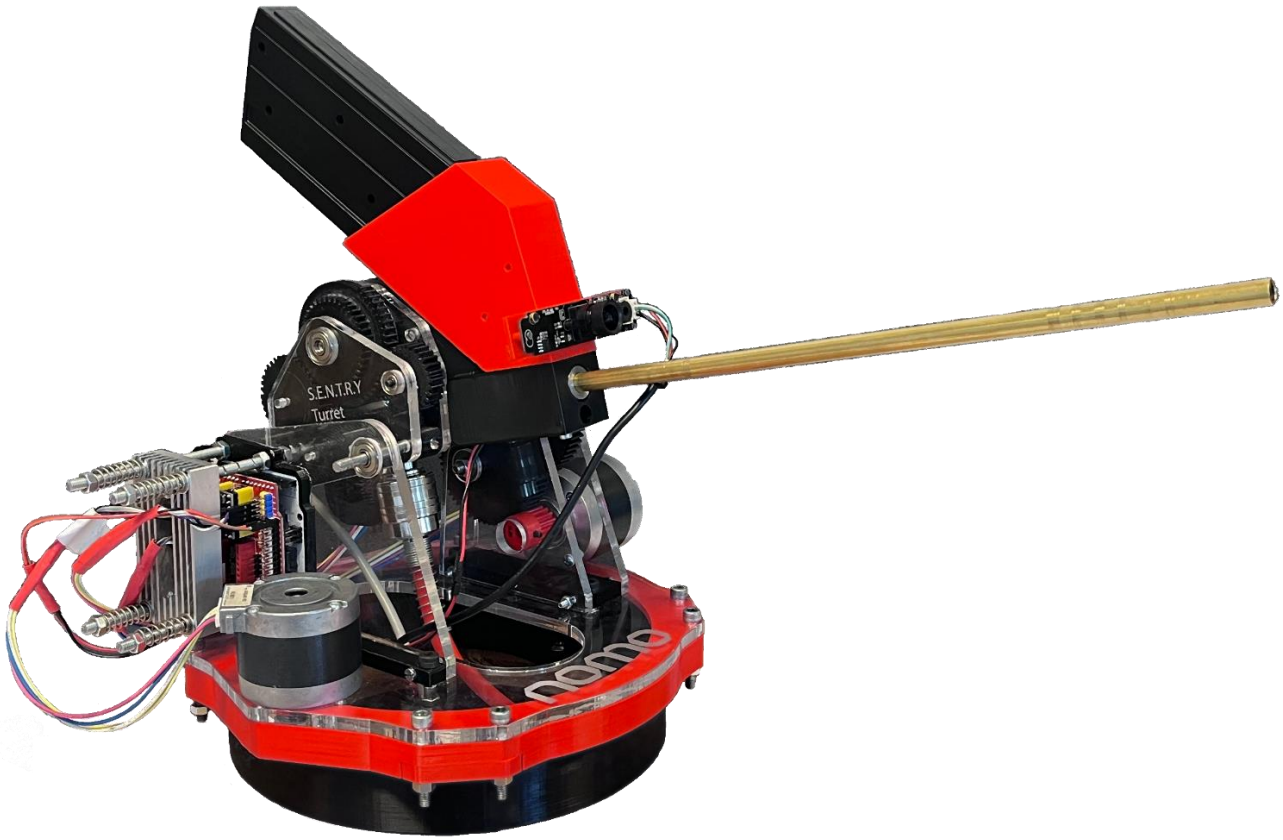
KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree project in Technology

First cycle 15 credits

Design and Construction of an Autonomous Sentry Turret Utilising Computer Vision

CARL BERMHED & JACOB HOLST



Stockholm, Sweden 2023

Design and Construction of an Autonomous Sentry Turret Utilising Computer Vision

Carl Bermhed & Jacob Holst

Bachelor’s Programme in Mechatronics

Date: 2023-06-06

Supervisor: Andreas Cronhjort

Course Leader: Daniel Frede

Examiner: Martin Grimheden

School of Engineering and Management

Swedish Title: Design och konstruktion av ett autonomt vakttorn nyttjande datorseende

TRITA-ITM-EX 2023:67

Abstract

The design and manufacture of a sentry gun turret capable of accurately and rapidly tracking and shooting moving targets was a challenging task that required delving into various engineering disciplines. This bachelor's thesis explores this challenge by presenting the process of construction, as well as the performance results of a turret created using a combination of 3D printing, laser cutting, and metal workshop manufacturing.

The subject was chosen to include different aspects of the engineering fields relating to mechatronics, and for the challenge of implementing and designing multiple systems that interconnect to effectively engage with a target.

The turret was created as a two-axis robot driven by stepper motors, with a gun driven by compressed air firing 6 mm plastic airsoft pellets using a clocked timing mechanism for rapid automatic firing. It was controlled by a system separated into two parts: a PC running facial recognition and colour identification software as well as performing movement calculations through python and an Arduino micro-computer running C++ controlling stepper motors and other hardware. The turret can accurately identify a target within five meters and with great speed home in and fire at the target with significant power.

Keywords

Mechatronics, Arduino, Computer Vision, Stepper Motor, Airsoft, Haar Cascade

Sammanfattning

Utvecklingen av ett autonomt vakttorn som med god precision och upprepbarhet kan hitta och följa ett mål var en utmanande uppgift som krävde användning av många delar av olika ingenjörsmässiga områden. Denna kandidatexamensuppsats utforskar utmaningen genom att presentera design och framtagningsprocessen samt redovisa prestandan av ett vakttorn tillverkat med 3D-utskrift, metallverkstadsmaskiner, och laserskärning.

Ämnet valdes för att inkludera olika fält i nära relation till mekatronik samt för utmaningen i att implementera och integrera elektriska och mekaniska system och på så sätt effektivt hitta och hantera ett mål.

Vakttornet är en tvåaxlig robot styrd med stegmotorer som har ett mekaniskt indexerad lufttrycksvapen som helautomatiskt avfyrar 6 mm airsoftkulor. Det mekaniska systemet kontrolleras av ett tvådelat kontrollsystem: en PC som kör ansiktsigenkänning och färgidentifiering i Python samt utför beräkningar för förflyttning utifrån kameradatan, samt en Arduino mikrodator med programvara i C++ som driver stegmotorerna utifrån förflyttningsinstruktionerna. Tornet kan med god precision identifiera, sikta mot och skjuta ett mål inom fem meter med en projektil av noterbar styrka.

Nyckelord

Mekatronik, Arduino, Datorseende, Stegmotor, Airsoft, Haar Cascade

Acknowledgments

A special thanks to our supervisor Andreas Cronhjort and course leader Daniel Frede for supporting us by answering all our questions regarding the project and sourcing the required components.

We would like to thank Isidor Lundqvist of Nomo Kullager AB for providing expert guidance on designing the slew ring bearing as well as providing us with ball bearings for the gearing and pitch axis.

We would also like to thank Jan Stammer and Mats Bejhem for their excellent advice in the design and manufacturing process of the many metal and plastic parts used in the gun assembly.

Finally, a thank you to our teaching assistants Omar Kanbour and Filip Strand as well as the other students working on their projects in the lab for showing great interest in our work, discussing, and sharing problems and solutions that helped us along the way and in general adding to a motivational working environment.

Stockholm, May 2023
Carl Bermhed, Jacob Holst

Table of contents

List of Figures

List of Tables

List of acronyms and abbreviations

MC	Micro Controller
MCU	Micro Controlling Unit
I/O	Input and Output
PC	Personal Computer
CPU	Central Processing Unit
FOV	Field Of View
HD	High Definition
CV	Computer Vison
AI	Artificial Intelligence
RGB	Red Green Blue
HSV	Hue Saturation Value
HC	Haar Cascade
UI	User Interface
PPS	Pulses Per Second

1 Introduction

This thesis describes and documents a project regarding the design, control, and construction of an automatic gun turret. It describes what technical solutions were implemented and why. It intends to share the results and method that was used to measure the performance of the turret and describe testing that can be used as a comparable evaluation between this and other similar projects.

1.1 Background

As the automation of tasks both physical and computer related continues to advance, the ability to interconnect systems that individually utilise different advancements in engineering is becoming ever more important.

One of the innate abilities advanced biological beings such as humans possess is identifying objects and interacting with them by moving its body through joints. These are individual fields of studies that have presented major advancements in the recent years, but there are still many applications that could benefit from the implementation of such an integrated system. Some applications for combining these different areas of study are the defence industry, transportation, security, and manufacturing. Humans excel at the reliability of these processes and the ability to adapt to error or new tasks but are lacking in accuracy and speed. For homogenous tasks that do not require adaptation to entirely new situations robots are far superior.

For the application of this thesis the ability to recognise a target from image input, driving motors to aim at a specific destination and firing a projectile at the target was required. These could in different ways be performed both automatically and by direct human control, but the automation of the entirety of the task would result in vastly improved performance. Some examples are automated anti-missile defence systems, automated fire extinguishing or automated local pesticide application for agricultural use, as well as many others.

1.2 Purpose

This project and thesis intends to document the design choices and technical solutions of an autonomous turret as well as investigate different fields of target identification, tracking and engagement. The research questions that were answered are:

- How can a robot identify and track a target?
- How can a robot achieve a balance of speed, accuracy, and reliability, and what trade-offs are necessary to maximise these factors?

1.3 Goals

Our aim was to build a sentry turret capable of finding a target using a camera, aim at the target's location by using motors and fire a projectile at it. To achieve this, the project was divided into three parts that interconnect, thus reducing the larger goal to one per part.

- Reliably find a target through a camera and relay instructions to the turret's motors in such a way that it aims at it.
- Aim at the target's location fast and reliably.
- Accurately fire a projectile at a target.

1.4 Research Methodology

This thesis is limited to the pure technical and mechanical aspects of the project, meaning that no ethical, political, or otherwise non-engineering discussion will take place. It is acknowledged that those discussions are important and relevant for this type of work, but they will not be included here.

The conducted project was focused on the design and construction of the turret and the challenges that had to be overcome, but the focus of the research is exclusively about the performance of the turret itself.

1.5 Delimitations

The scope of the project includes the combination of the three aspects: target identification, targeting and target engagement, the main delimitation is that these aspects will not be rigorously examined on an individual basis but rather as an integrated system.

The other noteworthy delimitations for the project are that all testing and performance was performed under reasonably normal conditions, for instance the targeting was tested in an environment where the target was something that distinctly separated it from the background and the sources of static and disruptions were low. The main effect of this is that no test of the robot's robustness to these conditions was assessed in this report.

2 Background

To support the project a foundation of both electrical and mechanical components, as well as logical principles must seamlessly interconnect. The more central or complex of these are described in this chapter.

2.1 Micro Controller

A micro controller (MC) or micro controller unit (MCU) is a small computer on a single integrated circuit that includes the major parts of a personal computer (PC) including a processor, input, and output (I/O) peripherals as well as both program and data memory. Program memory stores the instructions that are to be carried out by the central processing unit (CPU). The data memory stores the currently processed data made from the instructions carried out by the CPU and are often related to the data collected from the input peripherals. [1]

MCs are common components in everything from smart lightbulbs and cars to manufacturing equipment and machines. Depending on the computing power requirement of the machine the size of the MC can vary greatly and in more complex systems multiple MCs in combination with a larger computer can be used. In the case of machines with multiple MCs, serial communication is often used to receive and transmit information between the systems.

A small but powerful MCU is the Arduino Uno as seen in **Figure 1**. It is often used by hobbyist and professionals alike for rapid prototyping due to its dynamic qualities, size and low power consumption and cost compared to a PC.

Figure 1 Arduino UNO [18] ***Arduino Store*** **2021**

2.2 Serial Communication

Serial communication is a way of sending and receiving data from one server or computer to another where a series of binary digits are transmitted via cable, Bluetooth or WIFI. To send information, a computer can't simply send an 8-bit number but rather requires a protocol for how information is sent and received. This protocol includes more than just data and is there to help the computers communicate. [2]

Some serial communication protocols are only capable of sending data while others are capable of both sending and receiving at the same time thus limiting the use in some cases. The data being transmitted is stored in a buffer or a bus of a limited size before it is read this causes a risk of data loss if the buffer is not emptied at a sufficient rate.

2.3 Camera

There are three key values of a camera that are important to image processing and video analysis regarding target identification. These are Field of view (FOV), capture and refresh rate as well as resolution, these can vary depending on the model of the camera and can give captured images vastly different characteristics.

FOV is the width and height the camera lens can capture in terms of angle, with a small FOV the camera can focus on a specific object and is useful for taking pictures of individual items while a large FOV allows the camera to see objects very far to the sides. FOV is often given to the camera operator as an angle in degrees.

Capture rate is the speed of the camera and image processing. A high capture rate allows for a potentially smoother viewing experience as more images are captured every second. A high capture rate also allows the computer to be more accurate in its calculations since it can access more recent information and perceive smaller changes. The capture rate is transferred to the computer and is equal to the maximum number of images that can be displayed per second, this is called framerate.

Resolution is the size of the image taken by the camera. As the resolution of a picture increases, more individual pixels are captured and thus the potential for more detail is increased, but a picture with more pixels also requires more storage space or memory on a computer [3].

2.4 Computer Vision

Computer Vision (CV) is a subgroup of artificial intelligence (AI) that focuses on a computer's ability to recognise patterns in pictures. A human can easily identify objects by vision and instantly recognise its nature, categorise, and separate it from its surroundings. But since computers don't process images as we do, identifying something as simple as a geometric shape or a line can be quite troublesome. [5]

To a computer, a picture can typically be described as a large matrix where each element consists of a vector with three values, one each for the colours red, green, and blue (RGB), the displayed colour is the combination of these three values, it can be visualised as a cube where the three colours represent linearly independent vectors providing a three-dimensional colour space as displayed in **Figure 2**.

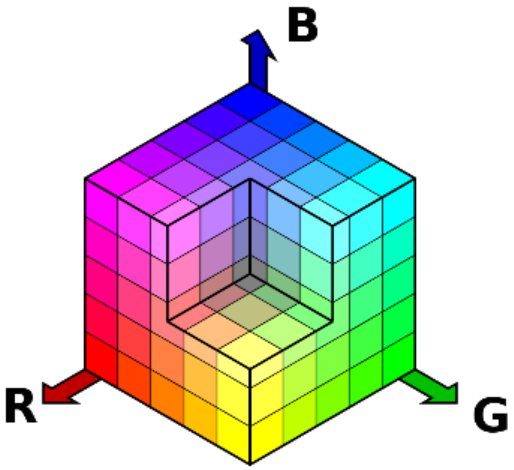


Figure 2, RGB Colour-Cube [19] ***SharkD (2023)***

The individual vectors are what we call a pixel. A computer capable of computer vision has through machine learning learned how to recognise patterns and guess what it is looking at. Some AIs can detect numbers or letters written on paper while others can detect faces and even individual persons if provided sufficient data and time.

OpenCV2 is a computer vision library in Python, Java and C and has many built in functions for both simple and advanced image processes [4]. The accuracy of these is proportional to the resolution of the image where a large number of pixels allows for more advanced processes like object identification, but it also results in heavier and slower computations due to an increased amount of data.

2.4.1 Image processing

When analysing a picture, a simple distinction that can be made is separating a different coloured object from the background, thus without using any computer vision reduce the space that needs to be analysed with simple operations. A select colour is chosen in any colour space and all pixels of that colour get selected for further processing.

However, the problem with this approach is that in the real world even an object that is perceived as having one colour is under natural conditions not illuminated evenly and thus from a computer's perspective consists of many different valued pixels. A span of accepted values can be created to improve this but an inherent problem with the RGB space is that colours that appear similar might have vastly different RGB values thus increasing the span may change the colours that pass the filter more than is desired or casing the limits to be unnecessarily complex.

In addition to the normal RGB matrix form, a picture can be described in a multitude of different colour spaces that have different strengths and weaknesses, and can be converted between these. [6] One of the colour spaces that coincide well with the human perspective is the hue saturation and value space (HSV), it can be visualised as a cylinder in **Figure 3,** with hue corresponding to what is interpreted as colour, saturation is the intensity and value is the brightness.

In contrast to the RGB space, the HSV space can house more accurate spans of colour at different light levels. Where the colour Is decided by the hue and a relatively small span is provided for it, while the saturation and value are allowed to have much greater spans whilst still preserving the inherent colour.

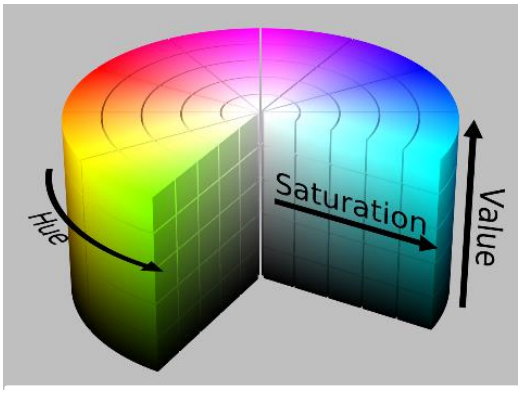


Figure 3, HSV Colour-Cylinder [20] *SharkD* (2023)

2.4.2 Mask Filtering

To simplify the vision aspect of the topic, the image in matrix form in the HSV colour space can then easily be processed to determine whether the values are within the specified range. This allows for a mask to be created filtering all values of the image to a binary form where 1 is within the specified values and 0 is not (often displayed as white and black respectively). This binary form allows for more complex processing such as contour finding, smoothing, sharpening and object mass identification to be performed through matrix convolution of an image matrix that is a third of the size of the original one. [7]

The image displayed in **Figure 4** shows the original image as well as a mask filter including all pixels with a hue of 200°±10°, a saturation between 20 and 99 and a value between 15 and 99.

Figure 4, Mask filtering Example

2.4.3 Haar Cascade

Haar cascade (HC) is an algorithm used in CV to detect objects in pictures and videos in real time. HC is an old but popular method of object detection and is one of the most popular object detection algorithms used in OpenCV2. The main advantage of HC is that it is fast enough to use with live video feed and simple to implement using the pretrained algorithms. The main disadvantage of HC is however that it finds many false positives in its object detection. It solves this by doing many different comparisons in a sequential order from easy to hard to easily skip those which are clearly wrong. [8]

2.5 Pressure Propelled Launching Method

The basics of pressure propulsion is imposing a high pressure behind a projectile that results in acceleration, as displayed in **Figure 5**. The blue section is of a higher pressure than the plain brass barrel. The source of this pressure is often an explosion but can come from principles such as compressing air with a piston or releasing gas from a pressurised tank. These methods have different aspects that make them suitable for different applications, but the principle remains the same. For the projectile to be able to use this pressure a barrel is needed to allow for the acceleration of the projectile over time, since when the projectile leaves the barrel the pressure around it becomes equal to the atmospheric pressure, the driving pressure is lost and the acceleration is halted.

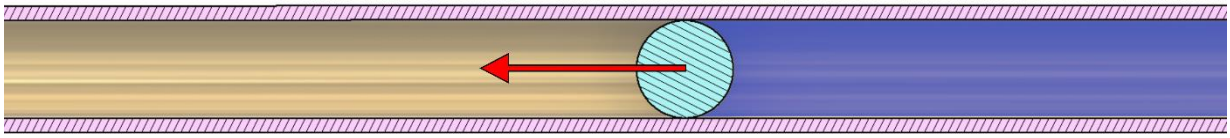


Figure 5, Pressure Propelled Projectile

As the projectile travels through the length of the barrel the volume behind increases and the pressure thus decreases, in addition to this some pressure is lost through heat dissipation and geometry. With a sufficiently long barrel the projectile would then experience deacceleration when the pressure has dropped enough. By this principle a length and pressure combination resulting in optimal projectile velocity can always be achieved.

2.6 Gearing

To transfer torque and rotational motion from the motors to the axes, many different variations of power transfer exist, gearing proves to be an ever-reliable variant that can easily be applied in most cases.

For gears both the module, “a” in **Figure 6** and face width are important factors that affect the longevity of the gears, since a larger module results in a larger gear tooth profile and in combination with face width reduces the stress that the gears suffer. The speed and torque transfer of different sized gears is directly proportional to the pitch diameter of the gears and thus proportional to the number of teeth. [9]

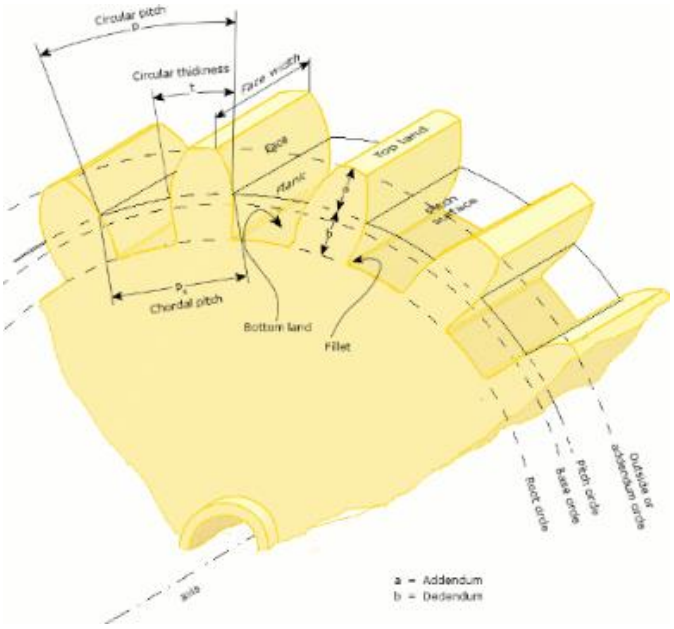


Figure 6, Gear Terminology [21]
Drawn by the uploading party. (2006)

2.7 Stepper Motors

In contrast to electrical motors that deliver continuous rotation, such as alternating- or direct current-motors and asynchronous motors, stepper motors magnetize poles of the stator in pairs as seen in **Figure 7**, causing discrete steps of movement of the rotor.

This results in high level of control and a holding torque. To achieve this however, some complexity is added to the system supplying the current. The motor's poles require discretised pulses of current in sinusoidal succession, something that is quite complex to achieve.

A simpler version that a lot of stepper motors use is discretised sinusoidal approximate pulses that result in a chopped version of rotation where each pulse results in a step, or a small rotation of the rotor. Sped up this results in what can be interpreted as continuous rotation. [10]

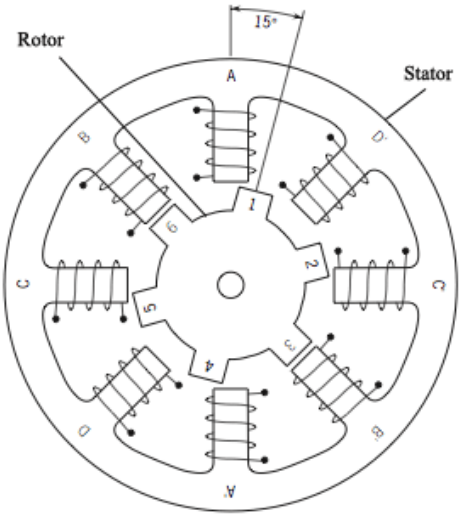


Figure 7, Stepper motor Internal Schematic [22] Circuit Digest (2018)

2.7.1 Stepper Motor Drivers

To enable use with an MC with limited current output, stepper driver units are usually used to allow an external power supply for the motors. In addition to this they also externalise the rotational instructions from discrete polarisation of the stator, resulting in a requirement of four in/out signals, pole A "+" and "–" as well as pole B "+" and "-".

The drivers simplify these to two to "step" and "direction" and converts that information in similar style to an H-bridge to deliver current to the correct poles at the correct timing. The DRV8825 stepper motor driver can be seen in **Figure 8** with the main chip in black and a small metal potentiometer used for limiting the current delivered to the motors.

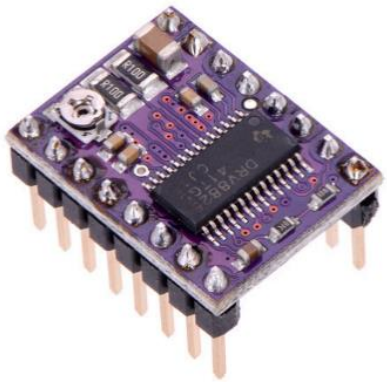


Figure 8, Stepper Motor Driver [23] Pololu (2023)

2.7.2 Stepper Motor Control and Microstepping

Stepper motors usually divide the full rotation of the axle into 200 steps, resulting in each step being 1.8°. Depending on the application this can be far too coarse and when run at low speeds it can become choppy.

To solve these problems most stepper drivers can perform what is called microstepping (MS), where multiple poles are partially magnetized, allowing for the step to be further divided, effectively halving the step size for each grade of MS. This provides a potential for higher precision and smother rotation but produces less torque since the poles are not fully energised. [11]

3 Method

To assess the research questions a two axial robot was constructed to test upon. To provide a basis of construction the method of testing is outlined in this chapter, the tests are intended as a general reference of evaluation.

3.1 System Scope

To reduce and concretise the scope of the project, requirements and delimitations were created to provide a basis of construction of comparable robots that will be able to be compared using the same tests that measure the performance of this project. It is noteworthy that the due to the nature of the problem, the delimitations that were chosen caused the results and conclusions of the thesis to be non-applicable for certain conditions.

3.1.1 Requirements

Reducing the purpose and goals to requirements the following was concluded:

- Ability to dynamically identify a target within a field of view of at least 70°.

The robot requires the ability to see a target within a reasonable field of view, and to be able to identify and separate the target from its surroundings.

- Ability to engage with a target within five meters of the robot's centre.

After identification, the robot needs to be able to move in such a way that the robot can engage with a target without hitting anything else in its surroundings. Due to the scope of the project, camera, and laboratory limitations, a distance of five meters was chosen for this requirement.

3.2 Fundamental Principle

The principal construction of the system was a three-part turret, where a baseplate was grounded in motion and securely fastened, a top portion was attached, rotating in the yaw axis, and a head was placed on the top portion allowing for movement of the pitch axis. These were controlled by stepper motors and received input from a camera mounted on the head in line in the yaw axis with the barrel.

The camera sent its images to an external PC that processed the image to instructions that were sent to the MC, which in turn transmited the signals to the motors.

The image processing that was relevant to the tests were constructed around the principle that a target was found on the image and the distance to the centre was calculated, this was then converted to motor steps that the MC can relay. It is noteworthy that due to physical limitations, the camera centre, and firing barrel do not coincide, this resulted in an error where the program does not aim the barrel but rather the camera.

If the target was sufficiently close to the centre, it was considered inside the dead-zone of the turret and no movement of the turret was commanded even if the centre was not entirely correct. This was to avoid small unnecessary movement since it would remain stationary until the target moved away from the centre, therefore leaving the dead-zone and being targeted using a larger movement.

3.3 Experiment Design

The turret had three primary parameters that could be tuned to change its behaviour. The delay, dead zone and aiming offset. The primary objective of the experiments described below was not to find the absolute limits of the turret or best performance in one individual aspect but rather to find the perfect balance between all the aspects of speed, accuracy, and reliability. The individual goal of each experiment and its purpose is described below.

3.3.1 Experiment Setup

For all experiments the initial robot and target setup were the same and as follows.

1. The robot was placed at a fixed spot in an environment of constant light and few visual disturbances.
2. The movement axes of the robot were placed in their idle positions.
3. A target was placed 5 meters away lining up with the centre of the visual sensor. (In the centre of the camera image)

3.3.2 Delay Experiment

The specific construction that was chosen for this project included both a PC and an MC that interacted via serial communication. Several errors could occur if the PC sends too many commands to the MC, if the serial buffer of the MC was full, no further commands could be received, and information would be lost. If the turret was currently moving and another command was sent the PC and MC would no longer agree on the current position of the turret, making it unstable and of risk of losing its target. Finally, some time was required for the MC to receive data and if too much data was sent it would take longer to process the data resulting in a longer delay until the turret moved again.

To prevent these problems a short delay was implemented. Too long of a delay would lower the speed and accuracy too much and with a delay that was too short, it would lose its stability. The delay contained two parts, a theoretical delay dependant on the physical rotational speed of the turret and a constant term that was added to provide a margin of error and to compensate for the serial communication itself. The experiment described below was to find this constant.

- A starting delay constant of 500 ms was set between each movement command. With the automatic targeting system enabled.
- The target was moved at a constant speed while the robot followed with its targeting system.
- If the turret managed to follow the target until it reached the end of its movement capabilities without losing stability, the test was repeated reducing the delay constant until the test failed.
- The optimal delay could then be concluded as the last delay that was stable.

3.3.3 Dead Zone Experiment

When the robot had found its target, it was susceptible to static that it perceived as small changes in the target location. It would always try to find the centre of the object, but a target always has an acceptable error where it would still be considered engaged with if hit. The dead-zone was the zone within the object that was considered within this tolerance.

A visual of a large dead zone is shown blue in **Figure 9**, the left picture shows the target outside the zone and the right has the target inside. To determine the size of this zone the following tests are conducted.

1. A dead zone of larger size than the target was set.
2. The targeting system was enabled.
3. The theoretical angle of movement from centre to edge of the target was calculated.
4. The target was moved slowly until it was just at the edge of the dead zone and the targeting system moved the turret head. It then reported the angle that it moved.
5. When the system the was still stable, the dead zone was reduced, and steps 2-4 were repeated until the stability decreased to much or the dead zone was sufficiently smaller than the target.
6. The final dead zone was chosen as the smallest that was still stable.

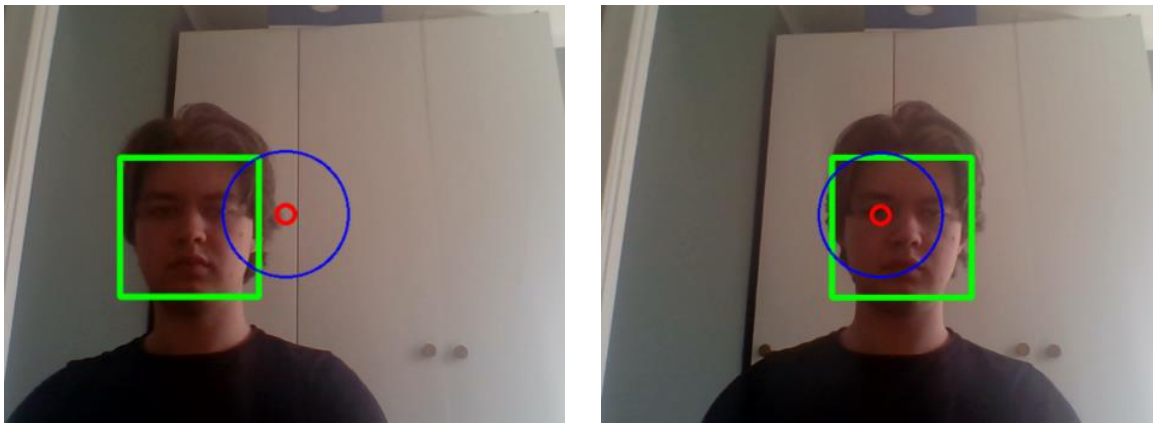


Figure 9, Picture of target Outside and Inside of Dead Zone (Blue)

3.3.4 Aiming Offset Experiment

To hit the target, the centre of the screen must line up with where the cannon fires its projectiles. To allow for an error the software was designed to allow for adjustment of the position it tries to centre the target to. This experiment was done to adjust the aim of the camera relative to the projectile line and the results of this test could either be used to adjust the camera position, barrel direction or to trim the software offset parameters. The test was complete when the adjustment done to the centre of the camera was equal to or less than one pixel in both the x and y direction, or the spread of the projectiles was so large that no further adjustment that improved the results could be done.

1. An accuracy evaluation target as seen in **Appendix A, Accuracy Evaluation Target.** was placed instead of a normal target.
2. A burst of five consecutive shots were fired at the target.
3. The distance to the centre in the horizontal, “x” and vertical, “y” directions were measured and the average position for each axis was calculated. In the case of major outliers these were removed from the calculation.
4. The software parameters were adjusted, and steps 1-3 are repeated until no further aiming improvements could be made.

3.3.5 Projectile Velocity Experiment

To assess the engagement of the turret, the projectile impact was chosen as a way of evaluation. To test this an experiment where the speed of the projectile launched was to be conducted.

1. The turret was placed close to a background of distinct colour relative to the projectile with incremental marks every 10 cm in the direction of the barrel.
2. A camera capable of high-speed footage was set up aiming perpendicularly toward the turret’s projectile path with a view of at least 130 cm.
3. A succession of shots was fired and filmed at pressures from 2 to 8 bars increasing by one bar each burst.
4. The footage was analysed, and the speed of the projectiles were determined according to **Equation 1**, where s is the distance travelled, **f** is the amount of frames that the distance is measured through and $\mathbf{f_c}$ is the capture rate of the camera.

$$v = \frac{s}{f * 1/f_c} \qquad \textbf{Equation 1}$$

5. The kinetic energy of the projectile is calculated per **Equation 2**where **E** is the total kinetic energy, **m** is the mass and **v** is the velocity of the projectile.

$$E = \frac{1}{2} * m * v\text{^}2 \qquad \textbf{Equation 2}$$

4 Design Approach

Reducing the problem to its most simple state, the robot required movement in two rotational axes with feedback to eliminate the error between where it wants to be and where it is. The mechanical principle was essentially just this, two axes that could be rotated with motors that ran on instructions from the feedback of a camera mounted on the construction that is moved with the axes. In addition to this an engagement mechanism of some sort was needed to provide feedback on whether the system was acting as it should or not. This could be several different things, such as a laser, light, radio or another kind of receiver or transmitter or in the case of this project a pressure driven gun.

4.1 Implemented Design

The implemented design separated the two axes into yaw, moving everything highlighted in green, and pitch, highlighted yellow in **Figure 10**.

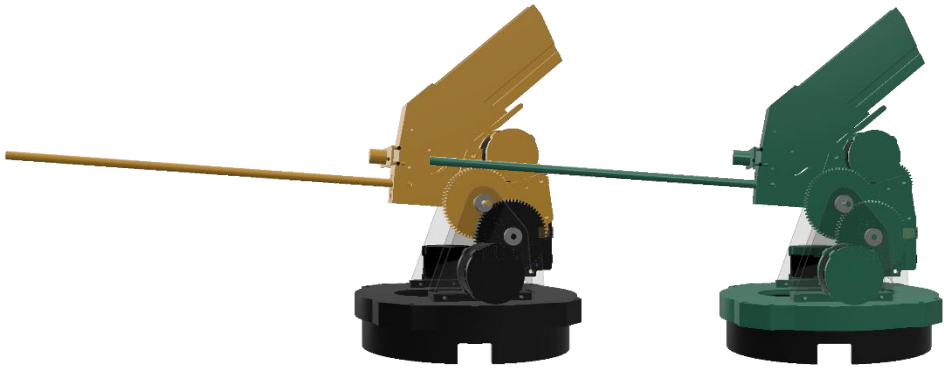


Figure 10, Top and Head Highlighted

The firing mechanism is mounted on the head and is mechanically clocked by timing gears that activate and deactivate reloading and airflow.

An overview of the entire system can be viewed in the flowchart in **Figure 11**.

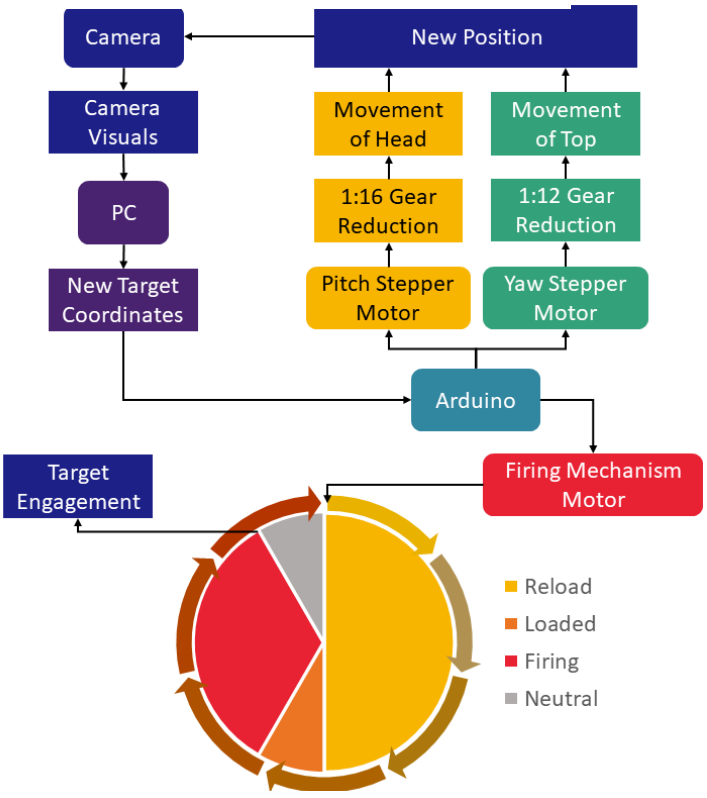


Figure 11 System Interaction Flowchart

4.2 Hardware Design

To support the project a physical robot was constructed, allowing for all requirements to be completed and providing a platform for the software to act upon the world with.

4.2.1 Pitch and Yaw

Designing the basic principle of the turret was done in a similar style to existing military tank gun turrets, particularly oscillating turrets such as the one in **Figure 12**.

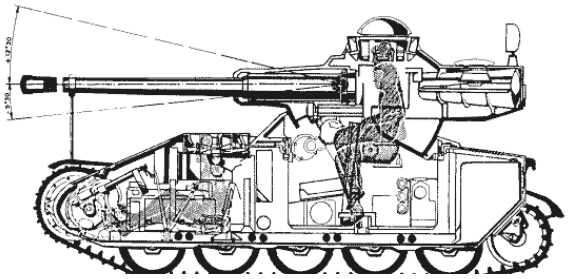


Figure 12, AMX-13 Oscillating Turret [24]
***The Modelling News* 2015**

In contrast to other turret designs oscillating turrets have their head fixed to the barrel, thus moving the entire top portion when changing the pitch and yaw. In a tank this results in some problems and difficulties with space and access that are not of concern for this project, but it simplifies and allows for the barrels centreline to intersect both the pitch and yaw axes of rotation at all times allowing for a theoretically simpler and more predictable angular movement as well as supporting the rather bulky firing mechanism in a compact manner.

In a similar manner as a tank the construction was bound to limits in the pitch axis, around -40° to +60° as well as allowing for at least three quarters of a rotation in the yaw axis.

To control the axes, KA50JM2-552 stepper motors were chosen since they allow for precise rotation, deliver a high holding torque, and offer sufficient torque at speed. **Figure 13**.

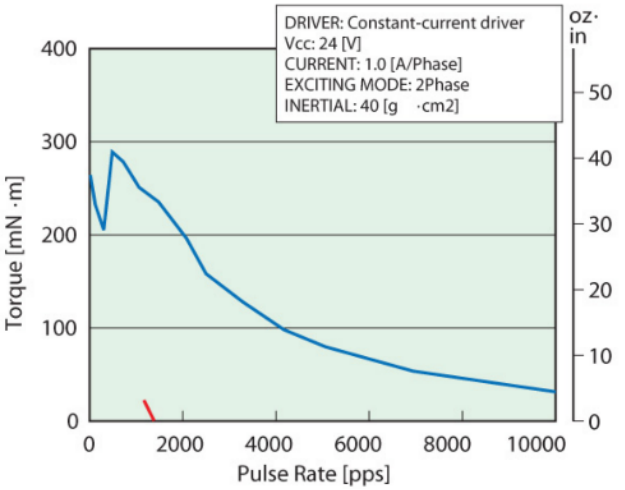


Figure 13, KA50JM2-552 ST TORQUE CURVE [25]
Elinco International JPC 2023

An acceptable torque was achieved at a pulse rate of up to 8000 pulses per second, thus allowing for a maximum motor speed of 2400 rpm delivering a torque of around 50 mNm and about five times as much torque at pulse rates below 2000 pps or 600 rpm.

4.2.2 Bearing and Slew Ring

The major forces that the structure must withstand was the weight of the construction as well as any loads applied firing the projectile and movement of the head. To allow for this the pitch axis must be supported by bearings that could withstand such forces, meaning that it must withstand both radial and axial forces in all possible directional combinations. In contrast the only significant load the pitch axis will suffer will be in the radial direction.

Considering these limitations, the head construction was designed to be able to be mounted on a simple axle supported by two ball bearings of the type: 625-Z that withstand the radial load and the torque that the head would produce. However, the pitch bearing had to allow for both wires and a hose for compressed air to be passed through the middle of the axis without tangling up. In addition to this, the scale of the construction would produce a great amount of torque if simply left supported by two standard ball bearings.

Taking this into consideration the implementation of a slew ring was the ideal choice. The slew ring itself was essentially a large diameter ball bearing as seen in **Figure 14**, using a gothic profile where four contact points always interacted with every ball locking the ball bearing geometrically as seen in Figure 15. This provided a preload that would otherwise have had to be provided externally. The bearings were separated by a bearing cage polyamide to prevent the ball bearings from colliding during rotation. This simplified the design significantly since one slew ring replaced two bearings. It also allowed the bearing to withstand loads in the radial direction as well as both axial directions and allowed the turret to be mounted upside down.

The slew ring had an internal ring gear integrated into its design to provide a direct linkage for motion at fast speeds and a proper amount of torque.

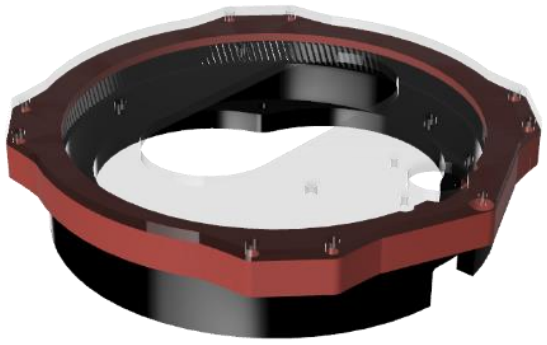


Figure 14, Slew-Gear Assembly

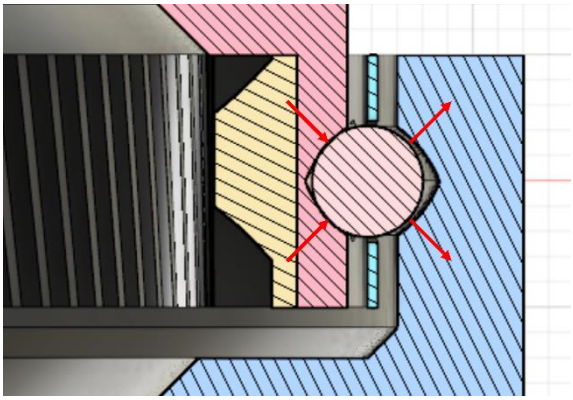


Figure 15, Section View of The Gothic Profile of The Bearing Assembly

4.2.3 Bearing Cage

A way to reduce friction and noise of ball bearings was to separate the balls with a cage, this was done using low friction materials, most commonly polyamides [12]. The separation of the ball bearings, in addition to reducing noise also reduced the number of ball bearings and guaranteed an even distribution throughout the ring. the bearing cage can be seen in yellow in **Figure 16**.

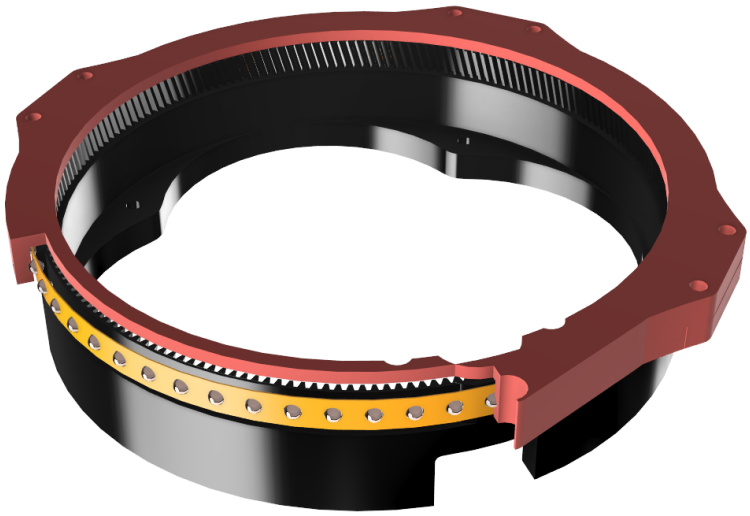


Figure 16, Bearing Cage

4.2.4 Motor gearing

To transfer the torque and rotation of the stepper motors to rotate the pitch and yaw axes a gear reduction was used. The motors could provide more than enough speed and since the angles the axes needed to reach were quite limited the speed at which the axes could rotate was allowed to be quite low relative to the motor speeds.

Geometrically it is advised to use gears with more than 11 teeth. [13] This, in combination with geometrical and calculational reasons resulted in the driving gear for both pitch and yaw were chosen to have 16 teeth and a module of 1mm.

The yaw gear and slew gear can be seen in **Figure 17**. For the yaw axis, the gears were chosen to have a helical profile to reduce noise and peak load of each gear tooth, similar results can be reached by changing the profile of the gear [14]. For the pitch axis the gearing used was however chosen to have a standard spur gear profile since the bearings could not properly withstand the axial loads created by helical gears.

Figure 17, Internal Ring Gear and Driving Gear

The pitch axis as seen in Figure 18 had a gearbox of a 16-tooth driving gear powering a 64-tooth gear coupled axially with a 16-tooth gear followed by another 64-tooth gear. This gearing provided a large gear reduction that allowed for proper torque and speed to be provided to the head. The full range of motion for the pitch and yaw axes were: 100° and 270° respectively.

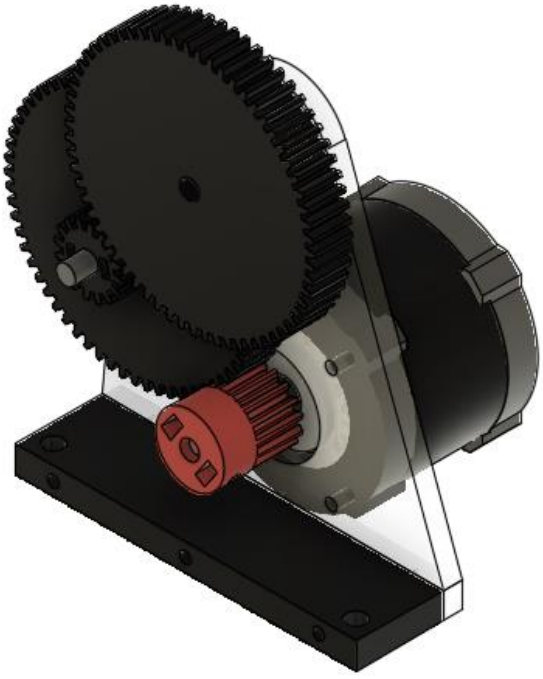


Figure 18, Pitch Gearbox and Motor

4.3 Projectile Launching

The implemented projectile propulsion system was directly based on using pressurized air as a propellant, this would allow for fast acceleration of the projectiles with high control of the exit speeds.

4.3.1 Pressurised air and pressure regulation

Due to the model being quite small, it was quickly concluded that mounting a full-size compressor on the model itself would not be viable. Instead, a smaller variant could theoretically be mounted on the turret. However, the solution that was chosen was to mount a compressor of the type Senco PC1010 externally and route its pressure hose through the centre of the yaw axis where it connected to the head. A pressure regulator was mounted before the hose resulting in the ability to change the pressure between 0 and 8 bars of overpressure by the turning of a regulator. The pressure was directly proportional to the exit velocity of the projectile, thus allowing for a regulation of the projectile speed and thus also the impact energy.

4.3.2 Air Pressure and Projectile Velocity

Using Bernoulli's equation, a plot between the supplied pressure and the exit velocity could be provided and as seen in **Figure 19**. The code generating this can be seen in **Appendix C, Matlab Velocity Plot**, notable is the flat line at 343 m/s, this is due to the fact that thermodynamical restrictions of a barrel of this type result in the inability to reach velocities above the speed of sound [15].

The plot does not factor in any friction from the barrel nor any change in pressure from the air supply, thus leading to an over estimation of the actual velocity.

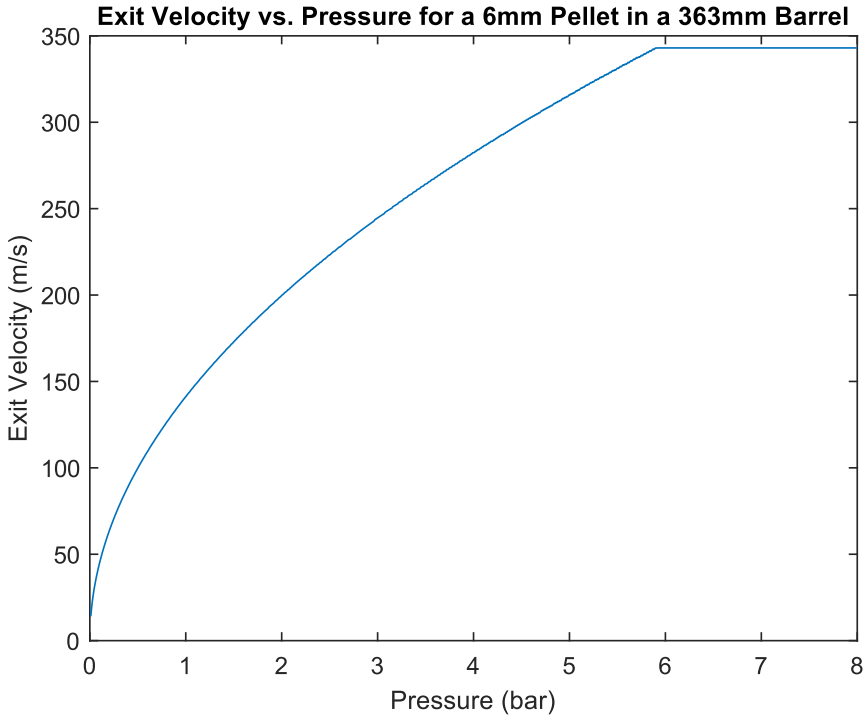


Figure 19 Exit Velocity vs. Pressure

4.3.3 Firing Mechanism

To launch projectiles, a fully automatic mechanism as seen in **Figure 20**, was implemented, driven by a motor where one rotation resulted in one cycle of the mechanism and thus a projectile launched. The mechanism had four phases as viewed in **Table 1**. A detailed picture set that displays these can be seen in **Appendix B, Firing Mechanism Cycle.**

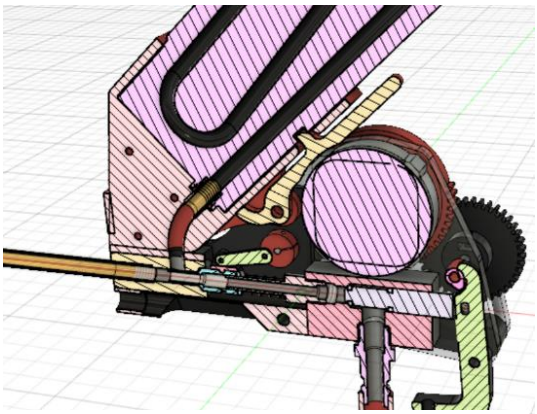


Figure 20, Projectile Launch Mechanism

Table 1, Projectile Launcher Gear Timing

Name	Timing %	Airflow	Reload
Neutral	8	Off	Inactive
Reloading	50	Off	Active
Loaded	8	Off	Inactive
Firing	34	On	Inactive

The motor was coupled to a timing gear with a diameter corresponding to 56 teeth, displayed in red in **Figure 21,** however only 28 were implemented, this resulted in the 28-tooth meshing gear of the reload mechanism completing an entire rotation for half the movement of the driving gear and then disengaging as to not complete more than one rotation.

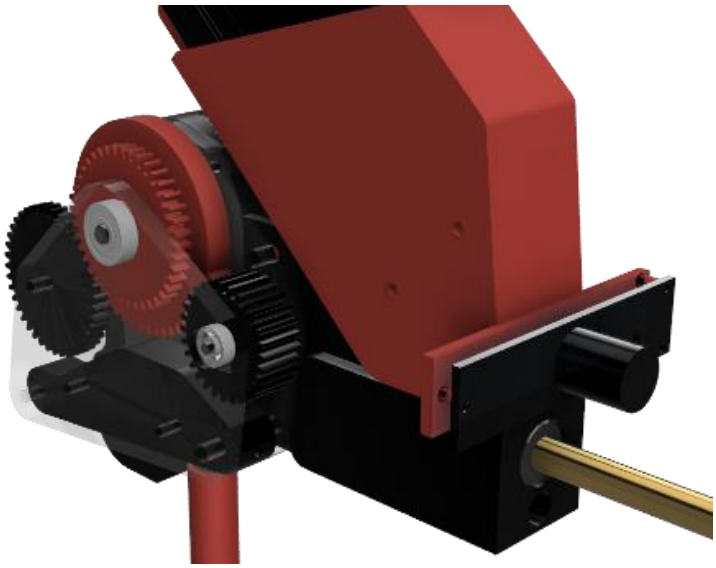


Figure 21, Timing Gear Assembly

The timing gear also had a 1:1 fully engaged coupling with a gear that drove a cam axle that released the pressure of the gun.

The timing gear activated the reloading and firing principles individually and at different intervals of the full rotation. This allowed the entire mechanism to cycle through the different stages safely restricted mechanically, thus hindering firing without having a projectile in the chamber or loading two projectiles without firing in between.

The driving motor of the mechanism was for practical reasons chosen as a stepper motor of the same variant as the pitch and yaw driving motors. It was however not necessary to use a stepper motor, and it may even have been preferable to use another kind of motor as very few of the specific functionalities of this stepper motor was used.

4.4 Software Design

To control the turret or for the turret to control itself some software was necessary. The program could be split into different parts, Target identification, instruction generation and conversion, data transfer and receiving, and motor movement. These are further explained in this chapter.

4.4.1 Separation of Software systems

The Arduino UNO has poor computing power compared to modern PCs, and a PC or laptop was not small enough to fit on the turret and could not easily drive stepper motors. To solve this the turrets software was split between the PC and the MC. The laptop, with more computing power handled the larger and harder decisions the turret had to make such as the image processing from the camera and the user interface (UI). While the Arduino only controlled the motors and mechanical components of the turret itself.

The added benefit of running the larger part of the program on the computer using python was that a lot of work had already been done providing libraries and resources such as the OpenCV2 library [4] made for image processing and Tkinter [16] used for the UI. The Arduino controlled the movement of the stepper motors using the Accelstepper and Multistepper libraries [17] for Arduino. Unlike DC motors the stepper motors required a signal for each step and could easily become unstable if the signals were not provided at the correct time intervals.

4.4.2 Interaction of Micro Controller and Image processing

The command sent to the Arduino was sent via the serial bus on the MC using an USB cable from the computer to the Arduino. Each command sent to the Arduino always had the same format, an example of a command sent is <0,200,200>. The <> brackets show the Arduino that this is the beginning and the end of a sent signal while the first number is a special command to the Arduino such as “Fire the gun” and the second and third to were the destinations of pitch and yaw steps that the motors should travel to. The code used for sending this data can be seen in **Appendix C, Python program for Turret Class, (Page XI)** in the “write_ard” function. On the receiving end the Arduino collects and parses the data in the “reciveWithStartEndMarkers” and “parseData” functions as seen in **Appendix C, Arduino Program for turret movement, (Page XIX-XX)**.

4.4.3 Instructional Conversion of Input

There were multiple ways to control the turret that were implemented through a user interface as seen in **Figure 22**.

All different control variations of the PC converted the instructions to absolute step positions that were sent to the Arduino. The step positions were calculated including all gear factors that were used and the proper variation of microstepping. To limit the turret within its pitch and yaw range a maximum and minimum value of degrees hindered the turret from moving to values beyond what was allowed. The limits were defined in **Appendix C, Python program for Turret Class (Page X-XI)** in the “__init__” function and used in the “limit_check” function **Appendix C, Python program for Turret Class (Page XIII)**

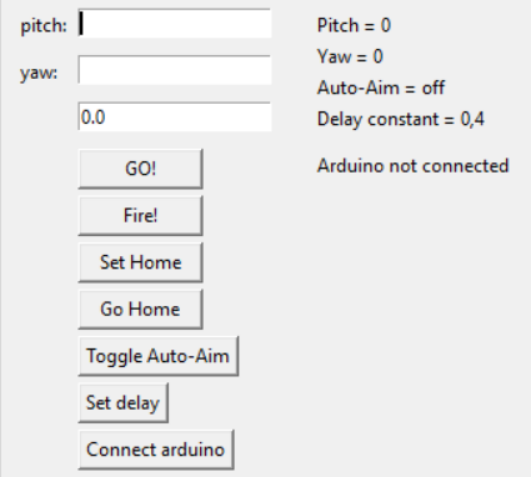


Figure 22 User interface and Tuning Operations

4.4.4 Text Input

The simplest form of instructional input was done by providing the PC with step locations that the motors should reach, via text input. As the user cannot input commands rapidly a delay was not necessary when sending the instructions to the Arduino.

4.4.5 Arrow Key Control

To dynamically test the movement of the turret, control via the PC keyboard keys was implemented in two variants, the first working in a similar fashion to the text input method of sending an exact location for the turret to reach, but with short step changes followed by a delay to gradually move the turret as an arrow key remains pressed.

The other variant was to send a command to the turret to the maximum position of the indicated direction of the pressed key. When the user let go of the key a second command was sent to the Arduino to stop the movement immediately regardless of whether the maximum position was reached or not. The latter version has a smoother and more predictable motion, but it is impossible for the computer to know where the turret was, therefore this version of the program was kept separate.

4.4.6 Pixel Selection

To intuitively control the turret using the video feed, the user could click on an object or pixel on the screen. When doing so the input produced a coordinate representing that point on the screen. This point was first converted from screen coordinates in x and y to an angle of the cameras FOV, the camera used for the turret had a FOV of ±42,5° and the resolution used was 960 x 720 causing an input of 480 on the far right of the screen to represent an angle of 42,5°.

The value in degrees was converted to motor steps, using both microstepping and the gear ratio, which in turn was sent to the Arduino in the same way as the text input. Mdeg and Mstep is the maximum degrees the turret was allowed to move and Mstep is that value in motor steps. SC is the max screen coordinate in that direction. The implementation of screen coordinate fetching is shown in the handle "handle_mousepress" function **Appendix C, Python program for turret main controls (Page VI-VII)**. The conversions were implemented in the "screentodegree" and "degreetostep" functions in **Appendix C, Python program for Turret Class (Page XIII)**.

$$deg = \frac{(FOV \cdot x)}{SC} \qquad \textbf{Equation 2}$$

$$step = \frac{(MStep \cdot deg)}{Mdeg} \qquad \textbf{Equation 3}$$

4.4.7 Colour Identification

The colour identification program used the HSV colour space of the image matrix from the video feed and filtered the values of each pixel to verify whether it was within a predetermined range, it then saved all pixels that passed the filter as a mask and highlighted all edges and compared all individual shapes by size to return the largest. The program then calculated the centre of the area of the largest target and compared it to the centre of the camera to generate movement instructions. Since this method received instructions from the computer continuously, a delay was implemented using the following equation. Where t is a constant addition that was determined by experimentation.

$$Delay = \frac{Steps}{Steps/Second} + t \qquad \textbf{Equation 4}$$

To tune what colour the user identifies a control panel as seen in **Figure 23**. was used in combination with the ability to click on a pixel and get the hue of it. The sliders allowed for specific and easy adjustment of the different parameters making colour identification possible. The main part of the colour identification algorithm is located in the **Appendix C, Python Code for Colour identification.**

Since the turret could only home in on one target at a time the largest object on the screen was chosen using the "select_largest_contour" function in **Appendix C, Python Code for Colour identification (Page XV)**.

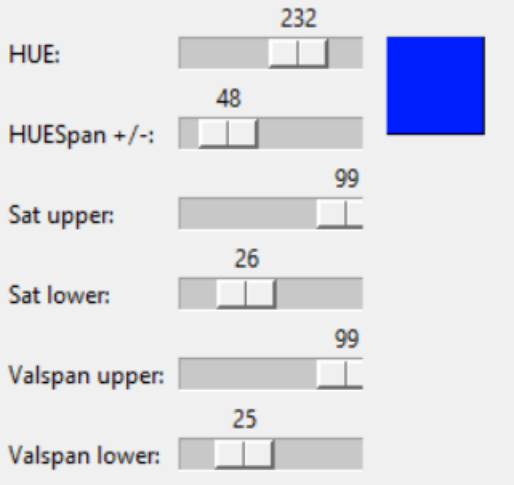


Figure 23, Colour Tuning Control Panel

4.4.8 Facial Recognition

To automatically find a target from the video-feed supplied by the camera mounted on the turret head, the external PC used python and ran a program using OpenCV2 and a pretrained Haar cascade to identify human faces. This part of the program is mainly located in the "find_target" function **Appendix C, Python program for turret main controls (Page II-III)**

The HC process required an image from the camera and returned a list of all probable faces including the position of the centre of the face inside the given image, as well as the width and height of the face. Instead of choosing the largest face, the program choose the face closest to the centre of the screen and calculated the distance, using the following formula where **x** and **y** are pixel coordinates with the origin at the centre of the camera. This is because the face closest to the centre is likely the face that was previously targeted and allows for the least number of movements.

$$\boldsymbol{d = \sqrt{x^2 + y^2}}$$

Equation 5

The image was also displayed in the UI so that the user could see where the turret was attempting to target with a rectangle displayed around the targeted face. This is visible in **Figure 24** where the green square is a found face and the red circle is the centre of the camera and picture.

To further reduce the processing requirements of the microcontroller and reduce unnecessary movement a dead zone of inactivity that was determined through experimentation was implemented in the image processing program.

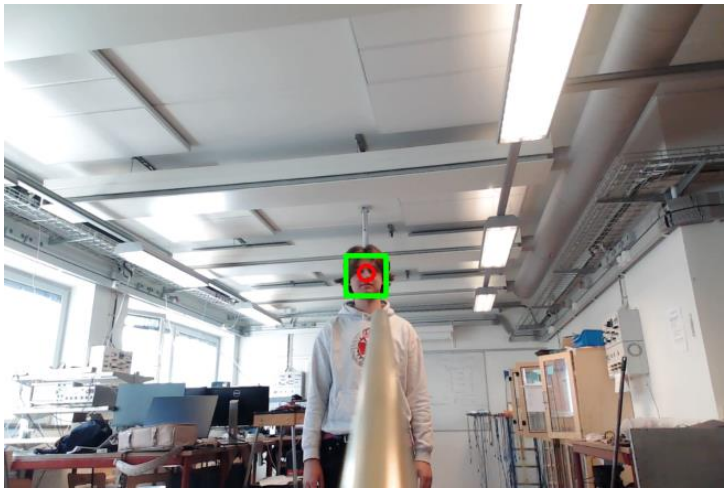


Figure 24, Target Found by Haar Cascade

5 Results and Analysis

This chapter presents both the results for tuning the turret parameters from the experiments that were described earlier as well as the overall results of the turrets key performance features.

5.1 Dead zone Experiment

The dead zone experiment was used to find the size of the area around the centre of the screen where the turret would not attempt to improve its position. This parameter affected both accuracy and reliability.

5.1.1 Results

After completing the experiment, a dead zone of 8 pixels was chosen, meaning that if the centre of the target was within 8 pixels of the centre of the gun, it would not try to home in on it. The reason for this was that any larger value resulted in the target being able to move further away from the centre of the screen than would be considered accurate without the turret correcting its position. Choosing a smaller value than 8 pixels resulted in the turret moving in tiny steps trying to correct errors that the rest of the construction could not support.

The dead zone was tested in increments of 5 pixels progressively lowered until the turret became unstable. This happened with a dead zone of 5 pixels and was clearly visible since small movements were made by the turret to improve its aim with no actual motion. After this the dead zone was increased incrementally by one pixel to 6, 7 and finally 8 where the instability was reduced to acceptable levels.

5.1.2 Reliability and Validity Analysis

The limiting factors of the test were not directly measurable, and it is noteworthy that due to limitations in testing resources a fully scientific and repeatable test could not be performed. The test does however prove useful due to the significant difference between a dead zone that was too small and one that was not.

5.2 Delay Experiment

The purpose of this experiment was to determine the delay constant t added when sending information from the computer to the MC. This parameter affected accuracy, speed, and reliability.

5.2.1 Results

The experiment concluded that a delay constant of 200 ms was the smallest stable value. This resulted in the turret being as fast as possible while still allowing for reliable tracking of targets. With this delay the turret did not lose control of its position and was able to follow a moving target with only slight trailing, and home in on the target as it stopped moving.

A larger delay resulted in the turret making larger moves but at a shorter interval which made the movements slower and the ability to follow the target with the same accuracy decreased. A constant delay of 150 ms made the turret erratic and unpredictable until it settled down and found stability. It was not able to follow the target with sufficient accuracy as the erratic moves disrupted the path. No test resulted in the turret completely losing control or losing sight of the target completely.

5.2.2 Reliability and Validity Analysis

The goal of the experiment was to find the shortest interval for sending data to the turret without it being overwhelmed and becoming unstable. Since the instability was rather binary in its form no further results than that delays below around 200ms would cause it to be unstable and delays above 200ms would be unnecessarily slow.

5.3 Accuracy Experiment

Two tests were conducted at different times, one directly after getting the turret to a functioning state, and one after operating it for some time.

5.3.1 Results of Test 1

The test was completed after five repetitions following the test protocol. A visual representation of the test can be viewed in **Figure 25**, Where all but the first burst and a shot from burst 4 is displayed. This is because they hit far away from the other shots and would reduce the visibility of the graph.

The detailed results of this test, including outliers can be viewed in **Appendix A, Firing Test 1**. In addition to this the outlier of burst 4 was not included in the adjustment calculation for burst 5.

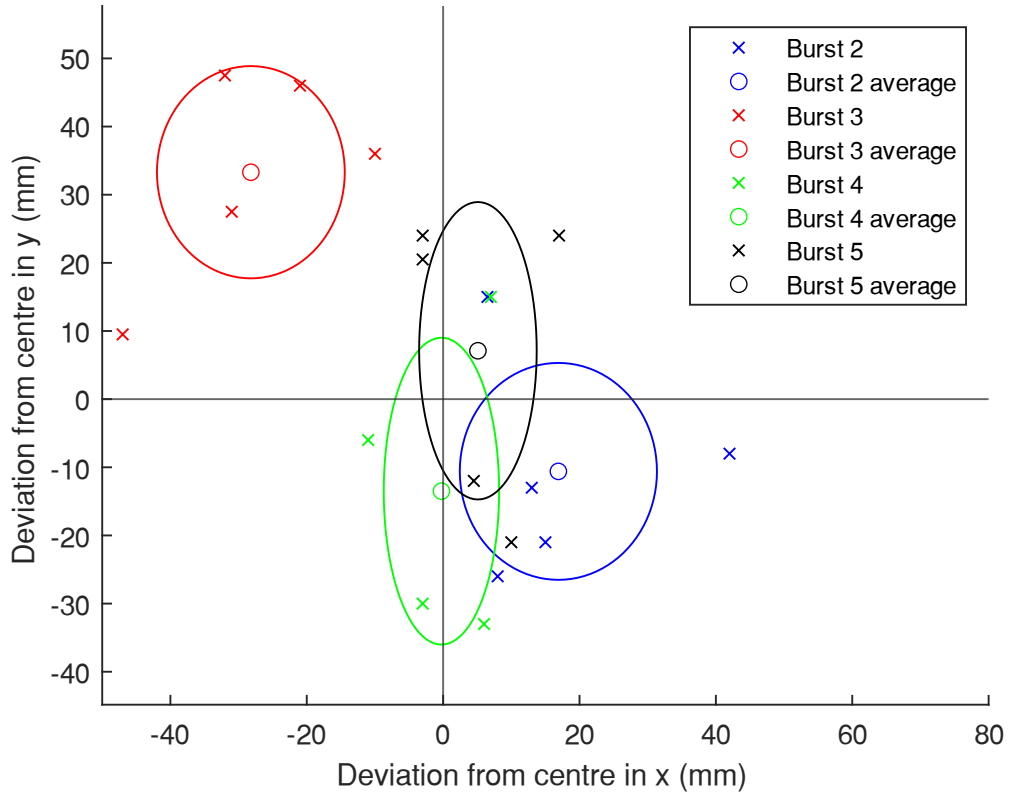


Figure 25, Firing Test 1

The ellipses display the standard deviation of the x and y axis, the ellipse axes in x and y direction each have a distance from the average point equal to one standard deviation of the axis. This shows an approximate hit zone.

5.3.2 Results of Test 2

The second test was completed after seven repetitions of the protocol, in the same fashion as test 1 and can be viewed in **Figure 26**, the major outliers were removed from the calculation of the following burst. The detailed results of each burst can be viewed in **Appendix A, Firing test 2.**

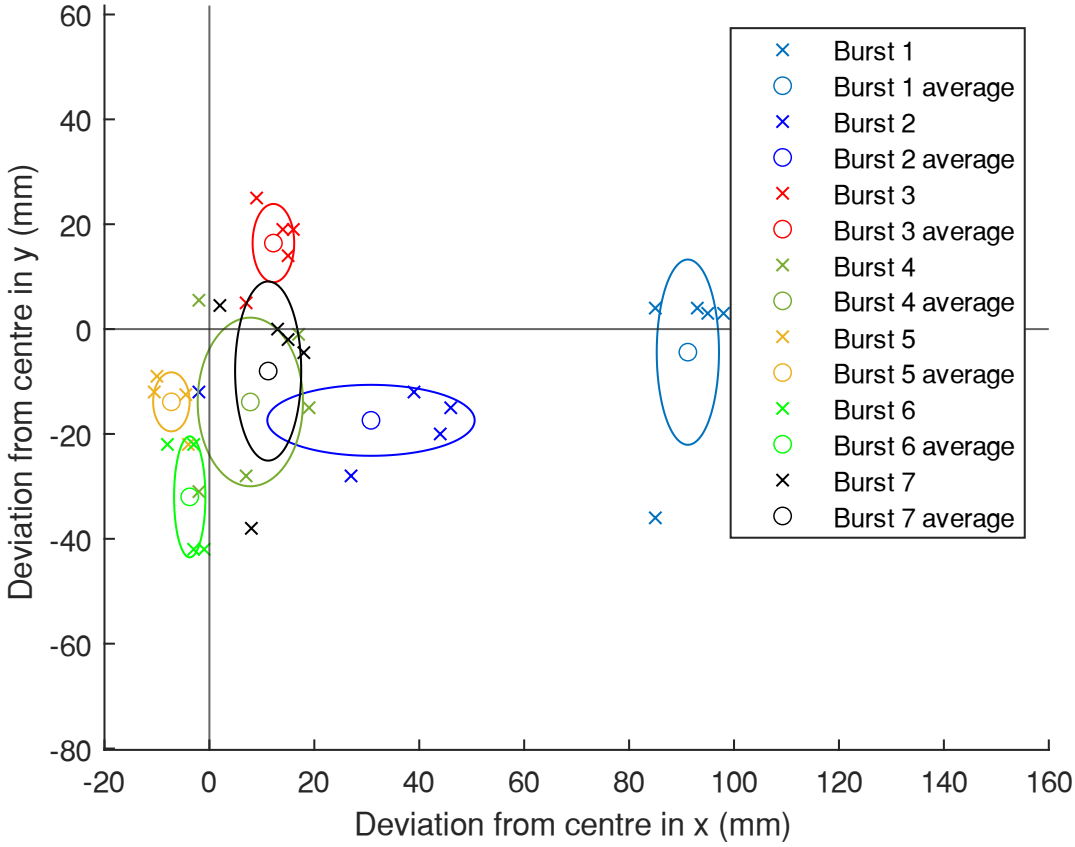


Figure 26, Firing Test 2

5.3.3 Reliability and Validity Analysis

It can be noted that with the current resolution the adjustments done at the end were of the magnitude of 1 pixel, this resulted in a decrease in accuracy due to rounding errors. In addition to this, the nature of randomised spread causes a test of only five shots to be limited in use for fine tuning.

5.4 Projectile Velocity Experiment

The test was conducted using a camera with a capture rate of 240 fps and in accordance with the test protocol beginning at two bars of pressure and increasing by 1 up to the maximum of 8 bars.

5.4.1 Results

The velocity calculated from the test can be viewed in **Figure 27**. Where the blue circles and red crosses are measured velocities of projectiles. The black line follows the average of each different set of velocities.

The total energy of a projectile with the average velocity for each pressure tested can be seen in **Table 2**, where it can be observed that at pressure above 4 bars the velocity increases significantly, and above 6 bars it began to decrease.

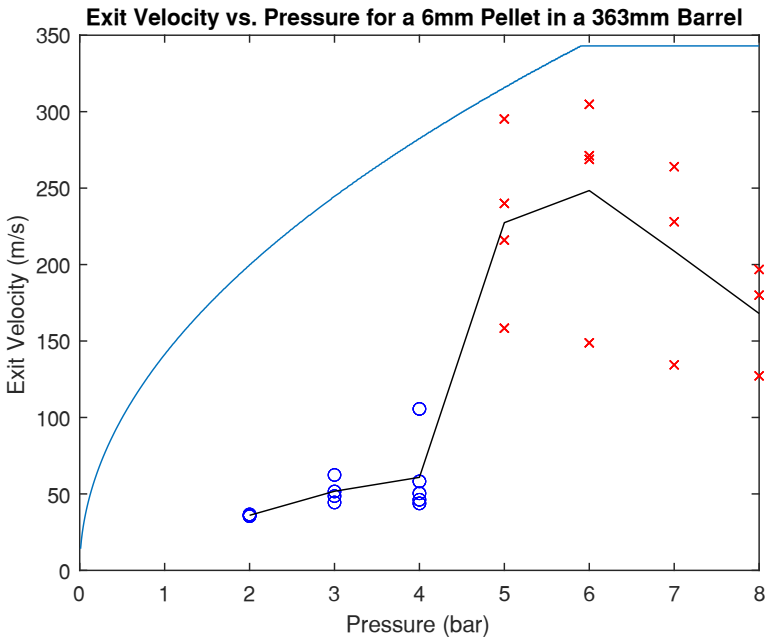


Figure 27, Projectile Velocity vs Pressure

5.4.2 Reliability and Validity Analysis

The test was somewhat restricted by the capture rate of the camera. At pressure above 4 bar, the projectile was only visible for one single frame, these were marked with a red X in **Figure 27**, the distance travelled between two frames could thus not be assessed and instead the distance travelled in the frame where it was visible was measured to the end of the barrel, this could result in the actual speed being measured as significantly less depending on the timing of the camera.

This impact could have been removed by using a camera with a higher capture rate or reduced by increasing the number of projectiles measured and removing outliers.

Table 2, Pressure, Velocity and Energy

Pressure (bar)	Average Velocity (m/s)	Energy (Joule)
2	36	0,13
3	53	0,27
4	61	0,37
5	227	5,17
6	248	6,17
7	209	4,36
8	168	2,82

5.5 Final Construction

The construction of the turret followed what is described in the thesis, however this was the result of an iterative process that resulted in three main prototypes.

In the first prototype, only the pitch and yaw axes were implemented since these provided a base testing platform for beginning work on the circuitry and coding. Prototype 1 as seen in in **Figure 28**, was the first physical variant that was functioning although it was not performing adequately on many different levels, primarily motionwise.

Figure 28, Prototype 1

The pitch motor produced adequate torque to move the relatively light head portion of Prototype 1, but it was found that if the weight increased this would proove troublesome and in addition to this the motors were not even close to their maximum performing speeds, meaning that the gear reduction could without compromise be increased.

In addition to this, the resistance of the yaw axis caused the motor to sieze. The first implementation of the slew ring bearing used 16mm glass balls and did not use any kind of separator, thus the balls collided with each other producing an unbearably loud noice.

When constructing the second prototype as seen in **Figure 29**, the gear ratios for both pitch and yaw were significantly increased and the implementation of a new slew ring gear featuring 4,5 mm steel ball bearings and a bearing cage of PLA provided a new foundation for the axes of rotation. These proved to be very viable for operating at the motor speeds that the motors performed well at and were finalised per **Table 3**.

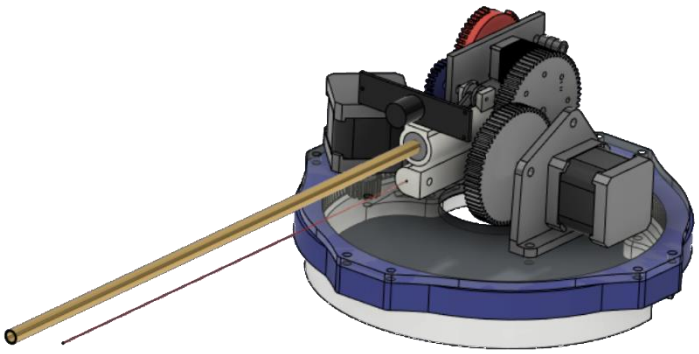


Figure 29, Prototype 2

Table 3, Gear Ratios and Motor Steps

Gearset	Helical?	Driving gear	Coupled Gear	Driven Gear	Final Ratio	Completed steps for full range	Time for full motion (seconds)
Yaw	Yes	16	-	192	1:12	1800	0,225
Pitch A	No	16	64	-	1:16	889	0,11
Pitch B	No	-	16	64			

This resulted in the theoretical full motion of the pitch to be reached in 0,11 seconds and yaw in 0,225 if the motors ran at a speed of 8000pps, and the turret being able to physically follow an object traveling at more than 75 m/s at a distance of five meters. These are however ideal values that do not factor in the inertia of the system nor the fact that such an object would not be picked up by the camera for enough frames. This shows that the theoretical speed of the turret was faster than needed.

A firing mechanism principle was also designed for this variant but was not constructed physically.

The final prototype as seen in **Figure 30**, had all systems fully implemented, the base of prototype 2 was principally the same with some minor changes of the yaw motor attachment point. When the firing mechanism was designed and constructed additional height was added to allow for the turret to aim further downwards. In addition to that a magazine holder and seat for the Arduino was included. Although the prototype still had much room for improvement it had the features and performance to perform the required tests. In addition to this the turret was mounted on a thick oak base to improve rigidity and reduce vibrations. Additional pictures of the finished turret with side by side views to the CAD variant can be viewed in **Appendix B, Views of the Completed Turret**.

Figure 30, Prototype 3

5.5.1 Performance Results

Some key attributes of the turret's performance were chosen to be analysed. These were the rate of fire, the speed of the yaw and pitch axes, the reaction speed, the targeting speed, the air pressure that the tests were performed at, the average velocity of the projectiles at that pressure, the maximum pressure that the gun could handle without performance loss as well as the average velocity of the projectiles at that speed. These parameters can be seen in **Table 4**.

Table 4, Turret Performance

Performance parameter	Value	Unit
Fire rate	6,25	shots/second
Yaw speed	75,4	degrees/second
Pitch speed	54,9	degrees/second
Reaction speed	53	milliseconds
Targeting speed	1100	milliseconds
Operating air pressure	4	bar
Projectile velocity at operating pressure	61	m/s
Maximum air pressure	6	bar
Projectile velocity at maximum pressure	248	m/s

The fire rate was determined by slowly increasing the speed of the motor powering the firing mechanism until it could no longer spin. The maximum speed was reached due to the stepper motor skipping steps and becoming unreliable.

The yaw and pitch speeds were determined by approximate testing of what rotational speeds the turret performed well at, these were not tuned in detail and could be increased at least fourfold but with a significant risk of physical damage to the construction due to abrupt acceleration. The values in **Table 4** show speeds at which the turret could comfortably move.

The reaction time is the duration it takes to send and receive a command from the user to the MC and was important to keep the turret ready, a reaction time of 53 ms is very low, making the robot vigilant for new commands.

The targeting speed was measured as the time for automatic targeting to find a target at the edge of the screen to the turret being dead centre on that target and ready to fire. This was the most critical value for the turret as it encompassed the entire performance of the robot. The statistics show that the turret could find a target and be ready to fire at it in just over one second finally proving that the turret was both fast and accurate.

The pressure of the air gun was evaluated based on the results of the velocity test. It was concluded that pressure up to 4 bars resulted in projectiles traveling at reliable speeds with no significant leakage of the mechanism. Pressure up to 6 bars was reliable and provided significantly faster velocities but it did not significantly improve the accuracy in a measurable way, since the gun was already accurate to within a pixel. To make the turret safer to be around an operating pressure of 4 bars was chosen. Nothing broke when operating at 8 bar, but significant leakage occurred which resulted in a lower performance with nothing gained.

6 Discussion

The overall performance of the turret was in accordance with the goals of the project and a good balance was found between the attributes fast, accurate and reliable thanks in large part to the parameters found through experimentation.

6.1 General Performance

A speed of 75,4°/s and 54,9°/s made the turret fast enough that it could follow a target at a reasonable pace and without moving so fast as to shake the turret apart. The fire rate of more than six shoots per second is comparable to airsoft guns and in combination with the reliability of the turrets aiming performance and firing velocity, is a significant way of target engagement.

It is also important to mention that the purpose of the three primary experiments conducted was not to evaluate the performance of the turret but to find the balance between the different tuneable parameters. The most important aspect of the turret was the balance between speed, accuracy and reliability and the absolute performance of the turret could always be improved by using different methods in both hardware and software.

6.2 Varied Dead Zone Performance

The dead zone experiment was based around the target being placed five meters from the turret, but since a target had the possibility of being anywhere in the range of zero and five meters, it was necessary that the dead zone also worked in between these distances.

Using the turret after the dead zone experiment took place showed that it was still applicable at these distances. However, since a small movement becomes amplified as the target moved closer to the camera the relatively small dead zone could not handle the larger motions. The dead zone of 8 pixels in radius was still applicable for the turret but a different approach might have been necessary to fully utilise this feature. It is possible that a distance dependent dead zone might have been able to compensate for this, at a distance the dead zone might be small to only compensate for the smallest of movements but larger when the target is closer to manage the amplification of these movements.

It is also important to mention that a dead zone of 8 pixels is not a universally true answer, this value can vary greatly on depending on primarily camera resolution, but also the mechanical performance in terms of microstepping, gear ratios, motor performance.

6.3 Serial Communication and Delay

As the communication between the computer and MC was vital for the performance, it was surprising that such a primitive solution as only sending data one way and relying on the accuracy of both machines gave such a good performance.

Since both the action of fetching and parsing data from the serial communication and the process of driving the stepper motors were active parts of the work for the MC it was hard to do both these things at the same time. The motors required constant pulling of the pins at precise intervals to move the motors at a constant speed and the communication had to regularly fetch new data to always stay up to date, if the pulling was interrupted by the acquiring of data the movement became uneven and unreliable. The usage of DC-motors or servos may have been much easier for the system to handle as they work more passively only requiring a PWM signal to work properly.

To improve the performance of the robot, the major improvement that would be necessary is a more robust way of communicating between the processing systems. This would ideally be two-way communication to avoid relying on the status of both machines matching in combination with a way of keeping the serial buffer free of data or reducing the two computing systems into one.

Much like the dead zone, the delay of 200 ms could also perform worse on shorter distances as the movements across the display were shorter at further distances than at distances closer to the camera, thus increasing the error of the delay. A distance dependent delay might have worked for this application. The delay might however not have been necessary at all with a more robust serial communication protocol.

6.4 Stepper Motors

Skipping steps was a common problem for the stepper motors that often went unnoticed during operation without any feedback implemented. It will made the turret drift to one side loosing what it previously considered it's home coordinates (0,0). Relying on the positional values of the two systems to coincide without any feedback was inherently flawed and this was most noticeable when the turret had been moving for a period of time with any of the automatic tracking variants enabled.

Restrictions of movement were made to reduce the risk of the turret damaging itself, and it was found that these values were reached before the actual positions were. This was clearly visible when running the home command, where the turret did in fact not return to its neutral position, but rather a position it had drifted to. This effect also seemed to be amplified when many small movements were attempted which in part was why the dead zone was implemented.

This did however have no effect on the turrets targeting capabilities as its output was always relative to the current position, but it broke the software limitations of the physical positions, resulting in movement that could damage the turrets more fragile components.

6.5 Fish-eye Lens Distortion

The camera used had a lens with fish-eye distortion, meaning that straight lines became bent when moving further away from the centre and objects become stretched at different angles of view. The result of this was that when the user or automatic targeting found a target and the program calculated the angle of that target it would be skewed slightly in relation to its real position. This resulted in the turret making movements that were slightly smaller or larger than what was accurate and therefore missing the target.

This was acceptable for the automatic targeting systems since it would soon make another move to correct itself from the error, but with manual point and click targeting the turret would often miss the intended location.

To combat this there are algorithms that could correct for this distortion which would straighten up bent lines digitally and would likely fix this problem. However, this process can be time consuming, complicated and will also result in a reduction of the image size. This was not considered during the project but is a possible improvement for future work.

6.6 Colour Picking and Tracking Algorithm

The colour picking algorithm is limited in its usability, for optimal performance an environment lacking the colour of the object to be tracked was required, this proved hard to supply and proves the real-world limitations of a pure colour based algorithm. Due to differences in lighting changing the values for "saturation" and "value for the colour it was difficult to implement a variant of the code where no undesired colours were selected if the target was not somewhere in the middle of the span and the background was not distinct.

In addition to this, an important restriction of the current code was that only one interval of colour could be selected. Due to the nature of the hue in the HSV colour format, red lies within an approximate interval of 0° to 30° as well as 330° to 360°. This resulted in red objects that were pure red being hard to track, due to different parts being in the different intervals while only one interval was searched for.

The static from the camera proved to be a source for unreliability of the picker tool, The colour that was selected could vary vastly. To combat this the tool was enhanced by selecting the average colour of the selected pixel as well as the pixels around it. This did decrease the probability of selecting incorrect colours but would require tuning to work properly, something that was outside the scope and timeframe of this project.

After the colour was selected the colour tracking worked as well as the face tracking variant with the exception of the selection of the largest object. The principle itself worked well, although because of the fish-eye effect objects closer to the centre appeared smaller than objects closer to the edge.

This caused the turret to home in on the visually largest object and in some cases when the sizes and distances were similar to another object within the colour span identify a new object closer to the edge as the largest one, home in on that one and oscillate between the two. This could be fixed by adding some other factor in combination with the largest object identification such as a distance to centre mapping like what was used for the facial recognition.

6.7 Motor Performance

To allow for precise movement stepper motors were used to turn the pitch and yaw axis. These motors performed well during normal operation but when running over an extended period of time produced a lot of heat. This caused them to perform sub-optimally and loosen the plastic gears attached to their axes, resulting in slipping and permanent deformation of the gears.

To keep the motors positionally reliable a delay was introduced to keep the movement steady. This resulted in a highly accurate robot, but it also resulted in many short steps being taken with abrupt starts and stops. This could in the long run put a lot of stress on the robot eventually causing components to loosen or break.

A more fluid motion would both function better and would put less stress on the gears due to less hard starts and stops. To achieve this two DC-motors could have been used instead, together with an entirely new control system for reaching a precise location. This would not have been impossible to implement to the current system but with some major changes to the physical model and software and it was thus outside the scope of the project. This is however a part of future work to make it even faster and more reliable.

6.8 Mechanical design

The mechanical design of the system was done in a successful way, both the pitch and the yaw axes moved in even and predictable ways due to the mechanical robustness of the system. Especially the slew ring gear proved to be effective and provided a strong foundation for the rest of the systems to be mounted upon. This effectiveness is due to the iterative design process where many variants of most parts was made. The first variants of the slew ring gear assembly were loud and bulky but were improved drastically in the later stages. The same principle was true for the head construction and motor mounts.

It is noteworthy that, even though they are not the parts that are most restrictive of accuracy or reliability, they are not free of backlash and play, meaning that extremely precise motion is not achievable in a repeatable way with the current tolerances of the construction.

6.9 Electrical Components

The least robust components of the construction proved to be the electrical components. Due to the large motors requiring a lot of power, the stepper motor drivers were prone to overheating, to combat this a large heatsink was created to dissipate the heat and prevent them from getting too hot.

6.10 Firing mechanism

Regarding the physical construction the firing mechanism was the most troublesome, even the final variant included some problems that it was designed to prevent. The main problem came with the precision of tolerances required for the sliding components, the axial alignment and concentricity required to seal air up to eight bar was not achieved properly.

The reloading system was expected to experience problems due to the inherent nature of many moving parts interacting but proved to be surprisingly robust. The timing mechanism was expected to experience some jamming problems when the timing gears started to mesh, since it was assumed that they would collide some of the times, but this was not as big of a problem as expected in part due to a slight modification of the gear profile increasing the pressure angle of one side of the gears. The only significant problem of the firing mechanism was that the motor, similar to the movement motors, overheated after prolonged use. This was detrimental to the mechanism as the timing gear could not receive torque from the motor and the mechanism could not spin. This was however easily fixed by replacing the worn gear. However the root cause of the overheating motors was not addressed.

6.11 Accuracy results

The gun proved to be highly accurate within the distance that was assessed. The final adjustments of the tests were within 1 pixel in yaw. A higher accuracy could be reached if the test was conducted using a higher resolution of the camera since the adjustments of the used resolution resulted in overcompensation.

In pitch the same principle could be seen in the first test but the second proved less reliable to adjust due to the turret having worn out its axis-gear couplings after prolonged use. However, the combined limitations of the control system and the pitch and yaw adjustments were significantly larger than the final adjustments of the test, proving that the firing mechanism was sufficiently accurate.

To confirm the viability of a shot hitting the target, a system similar to the dead zone could be used where the software could provide a percentage chance of hitting the target based on how much area of the target is within the expected “hit zone”. Since the spread of the turret gun is probabilistic the hit zone would have to make use of aspects such as standard deviation to calculate the chance of hitting the target. In this example it would also be highly appropriate to improve the accuracy of the gun by replacing the barrel with a precision bored and rifled barrel to introduce stabilisation spin to the projectile.

6.12 Velocity Results

The Velocity of the projectiles increased in line with the supplied pressure, not in accordance with the simulated values however as seen in **Figure 27, Projectile Velocity vs Pressure**, at lower pressure the velocities were significantly lower than predicted. Even though significantly higher, the velocities at higher pressure never reached the corresponding calculated velocities.

The first principle that caused this was the static pressure loss of the mechanism itself that was not included in the calculations, when directing fluids through restrictions the pressure drops due to the geometry of the construction. This would cause a loss that is somewhat static in nature and result in lower pressure by a static factor.

A significant increase in velocity was noted between four and five bars of pressure, although not confirmed this was theorised to either be because of some resistance in the barrel acting upon the projectile and air at lower pressures, or because of the flow in the pipes becoming turbulent at the higher pressure causing a significant jump between 4 and 5 bars.

In addition to this a drop in velocity was noted above 6 bars, this was due to excessive leakage of the gun, although not measured the pressure of the compressor quickly fell to a more stable point when exceeding the pressure threshold of the gun.

The uncertainty of the velocity increased significantly at higher pressure since the projectile could only be captured on one frame of camera footage. This can be seen as a significantly larger spread of speed at those pressures as the cameras timing captured the projectile at vastly different locations. This would however never increase the speed measured, and thus the top speeds would still be considered accurate.

6.13 Projectile launcher Improvements

To improve the performance of accuracy, precision, and velocity of the projectiles a more proper barrel could be mounted on the turret, ideally one with rifling to provide spin for stabilisation. This could in theory increase these factors significantly. This improvement would however not be measurable without an increase in resolution of the camera. The current resolution of 960 x 720 was not the maximum of the camera that was used, that being 1920 x 1080. This increase would provide significant tuning capability improvements.

If the turret was to be upscaled for higher distances the resolution would have to be orders of magnitude higher to be accurate, something that is not feasible. Instead, the addition of a camera with similar high resolution but with a lower FOV, such as a telephoto camera could be used in addition to the wide-angle camera. This would allow the wide-angle camera to identify and home in until the target was within the FOV of the telephoto camera and use it to home in more accurately. In addition to this the combined pictures of the camera could be merged to provide a wide-angle image with a high resolution in the centre.

An alternative would also be to implement a camera with a mechanically controlled optical zoom function to achieve the same functionality using only a single camera. An implementation of a camera capable of optical zoom could be used to create a dynamic motion control where the movement of the turret is proportional to the FOV of the camera instead of specific degrees of rotation. This would enable the turret to rotate at higher speed but with reduced precision when zoomed out and slower and with enhanced precision when zoomed in. If using stepper motors with this setup the micro stepping amount could be increased with the zoom allowing for extremely smooth and precise movement without sacrificing performance

7 Conclusions and Future work

This section focuses on the entirety of the project, reflecting on the work that was done and highlight a path for future work.

7.1 Conclusions

The overall experience of the project has been highly positive, and the results reflect this. The turret had a robust construction capable of tracking a moving target and despite being constructed mainly of plastic it is surprisingly sturdy and capable of being oriented in many ways, making it highly versatile.

The stepper motors allowed for precise movement despite some skipping of steps and made the turret highly accurate and fast. The program that was running on the MC was rudimentary but capable of steering the robot and the gun whilst also remaining safe and unlikely to break. The PC ran the main program efficiently utilising its substantial computational power to alleviate the MC of any unnecessary computing.

It had a highly capable and adjustable UI for the user to control the robot through and a very effective targeting system capable of autonomously finding and tracking faces and separate coloured objects from their surroundings. The turrets main program and class worked fast and efficiently and had multiple layers of security to prevent the robot from breaking itself.

There are also multiple opportunities to improve the performance of the turret mainly introducing new protocols for both input and output data, read more of this in **7.3 Future work** and previously in **6 Discussion**.

A large part of the project was to engage with multiple different fields of engineering, but this also meant to integrate these fields which was a central challenge throughout the project. The project also became a viable platform to test multiple approaches to the same problems, both for the current construction of the project as well as acting as a base of comparison for similar projects.

As the different systems of the turret met the goals of the project, redoing the project, the focus would likely be on either improving the smaller aspects of the individual systems or trying different methods as described in **6 Discussion**. Another approach would be to completely rethink the construction and trying some of the points in future work that would change the system fundamentally.

7.2 Limitations

The main limitation of doing the main part of the computing on the computer and sending instructions to the Arduino was the speed at which this was possible, it forced the turret to operate at a slower pace than necessary and made it less reliable primarily regarding its own locational information. The alternative would have been to perform all the processing and movement controls on one computer. This could have been achieved with a small formfactor computer such as the Raspberry Pi and would likely make the turret operate faster in certain aspects but much slower in others like the image processing.

7.3 Future work

There are a lot of possibilities for future work for improving the turrets performance, building on the discussion points in **6 Discussion** the following improvement points of varying magnitude are proposed.

- Use a more robust serial communications protocol.
- Create a dynamic dead zone.
- Introduce a hit zone for accuracy verification.
- Improve the delay function.
- Adjust for fish-eye lens distortion limitations.
- Create a more robust colour picking process.

Some areas also have the possibility to be completely reimagined for a potential increase in performance.

- Replace the stepper-motors with DC-motors and a control system for a more fluid motion.
- Join the two programs on the computer and the MC to one unit, to eliminate serial communication.
- Add zooming functionality to the camera and change the control system to dynamically adjust the rotational resolution.

7.4 Reflections

Automation software and facial recognition can be used for a variety of different purposes, some of which are useful and productive and other are malicious to say the least. The combination of facial recognition, automatic targeting and automatic firearms is an area of great ethical discussion and even though it has not been discussed in the report, it has always been in mind during this project. Great care has been taken as to not hurt anyone and no automatic firing without human input was ever implemented because of the risks associated with combining these aspects. This project has been made purely for academic purposes and we distance ourselves from and do not consent to this project being used for purposes that could result in human harm.

8 References

[1] Techtarget, “Microcontroler (MCU),” 2019. [Online]. Available: https://www.techtarget.com/iotagenda/definition/microcontroller. [Accessed 16 April 2023].

[2] Swaroop, “What is Serial Comunication and How it Works?,” Codrey, 2018. [Online]. Available: https://www.codrey.com/embedded-systems/serial-communication-basics. [Accessed 17 April 2023].

[3] Adobe, “Everything you need to know about image resolution,” [Online]. Available: https://www.adobe.com/uk/creativecloud/photography/discover/image-resolution.html . [Accessed 18 April 2023].

[4] Open CV, [Online]. Available: https://docs.opencv.org/4.x/index.html. [Accessed 18 April 2023].

[5] IBM, “What is computer vision?,” [Online]. Available: https://www.ibm.com/se-en/topics/computer-vision. [Accessed 8 April 2023].

[6] Vocal, “RGB and HSV/HSI/HSL Color Space Conversion,” VOCAL, [Online]. Available: https://vocal.com/video/rgb-and-hsvhsihsl-color-space-conversion/ . [Accessed 9 March 2023].

[7] Stanford, “Tutorial 1: Image Filtering,” Stanford University, [Online]. Available: https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html. [Accessed 16 April 2023].

[8] A. Jaiswal, “Object Detection Using Haar Cascade: OpenCV,” Analytics Vidhya, [Online]. Available: https://www.analyticsvidhya.com/blog/2022/04/object-detection-using-haar-cascade-opencv/#:~:text=What%20are%20Haar%20Cascades%3F,%2C%20buildings%2C%20fruits%2C%20etc. [Accessed 16 April 2023].

[9] in *Maskinelement Handbok*, Instutionen för maskinkonstruktion Kungl Tekniska Högskolan, 2008, p. 49.

[10] Dejan, “How to Control a Stepper Motor with A4988 Driver and Arduino,” How to mechatronics, [Online]. Available: https://howtomechatronics.com/tutorials/arduino/how-to-control-stepper-motor-with-a4988-driver-and-arduino/ . [Accessed 16 April 2023].

[11] D. Collins, “What is microstepping,” Linear motion Tips, [Online]. Available: https://www.linearmotiontips.com/microstepping-basics/. [Accessed 16 April 2023].

[12] SKF, “SKF,” SKF, [Online]. Available: https://www.skf.com/group/products/rolling-bearings/principles-of-rolling-bearing-selection/bearing-selection-process/bearing-execution/cages. [Accessed 25 05 2023].

[13] J. Huo and H. Zheng, “Gears wiht the minimum number of teeth and high contact ration”. World Intellectual Property Organization Patent WO2004020875A1, 2002.

[14] M. Åkerblom, “GEAR NOISE AND VIBRATION-A LITERATURE SURVEY,” Volvo Construction Equipment Components AB, 2008.

[15] J. D. A. Jr., Fundamentals of Aerodynamics 5th ed, McGraw-Hill Education, 2011.

[16] The Python Software Foundation, “Python interface to Tcl/Tk,” [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed 16 April 2023].

[17] M. McCauley, “AccelStepper library for Arduino,” Airspayce, [Online]. Available: https://www.airspayce.com/mikem/arduino/AccelStepper/. [Accessed 16 April 2023].

[18] Ardiuno Store, “Arduino UNO Rev3,” [Online]. Available: https://store.arduino.cc/products/arduino-uno-rev3. [Accessed 17 April 2023].

[19] SharkD, “RGB color model,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Color_model#/media/File:RGBCube_a.svg. [Accessed 9 April 2023].

[20] SharkD, “HSV cylinder,” Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSV_color_solid_cylinder_saturation_gray.png. [Accessed 9 April 2023].

[21] Drawn by the uploading party, “Gear Words,” Wikipedia, [Online]. Available: https://commons.wikimedia.org/wiki/File:Gear_words.png . [Accessed 17 April 2023].

[22] Circuit Digest, “Introduction to Stepper Motors,” [Online]. Available: https://circuitdigest.com/tutorial/what-is-stepper-motor-and-how-it-works. [Accessed 17 April 2023].

[23] Pololu, “DRV8825 Stepper Motor Driver Carrier,” [Online]. Available: https://www.pololu.com/product/2982. [Accessed 17 April 2023].

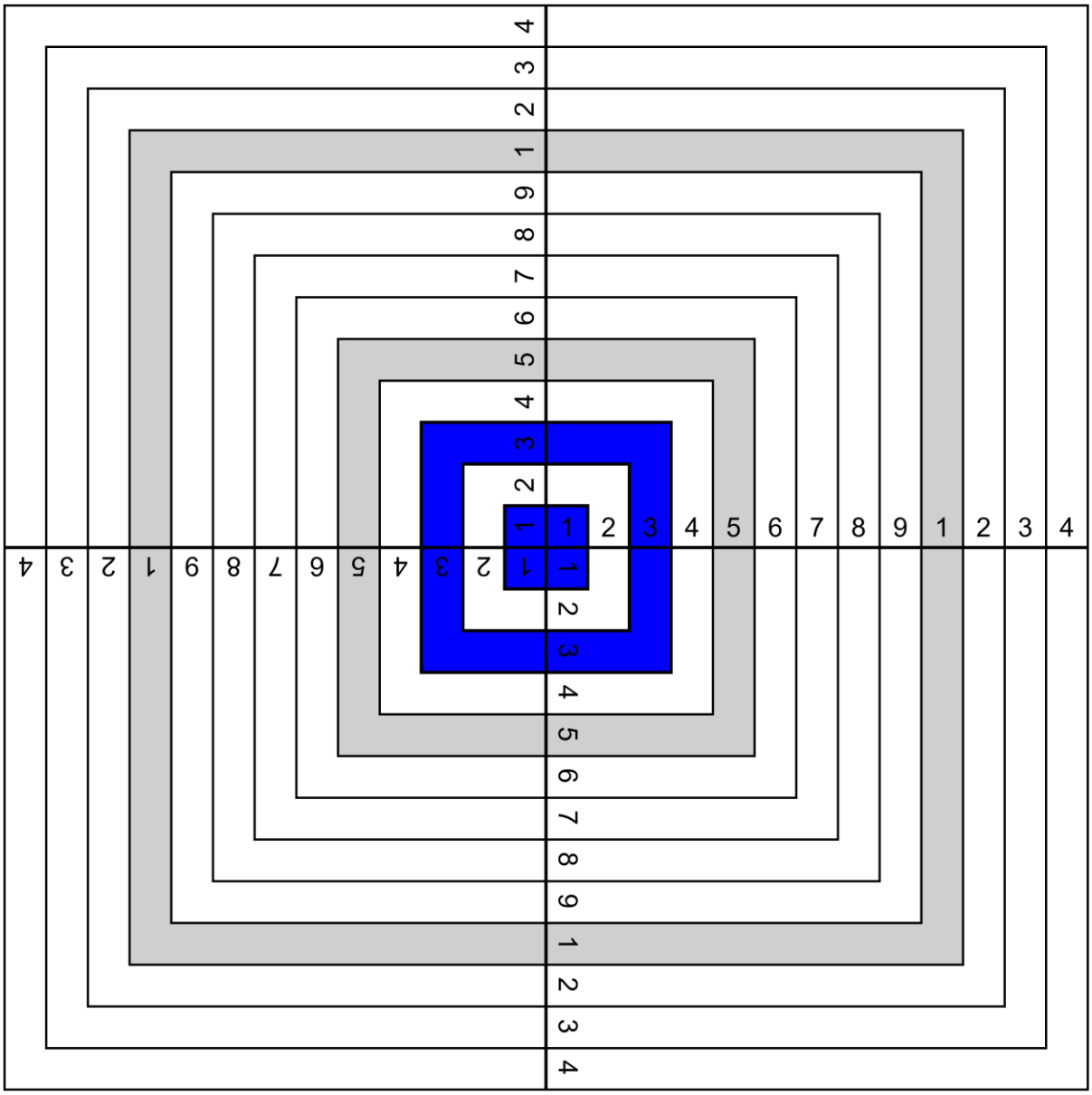
[24] The Modelling News, “AMX-13[Technical Drawing,” The modeling news, [Online]. Available: https://www.themodellingnews.com/2015/11/takom-new-amx-13-series-3-french-tanks.html. [Accessed 16 April 2023].

[25] Elinco International JPC, “ITEM # KA50JM2-552, KA50 SERIES 2 PHASE HYBRID MOTOR (1.8 DEGREE/STEP) - NEMA 17 [Product data sheet],” Elinco International JPC Precision Rotating Components, [Online]. Available: https://catalog.e-jpc.com/item/stepping-motors/2-phase-round-ka-series/ka50jm2-552. [Accessed 16 April 2023].

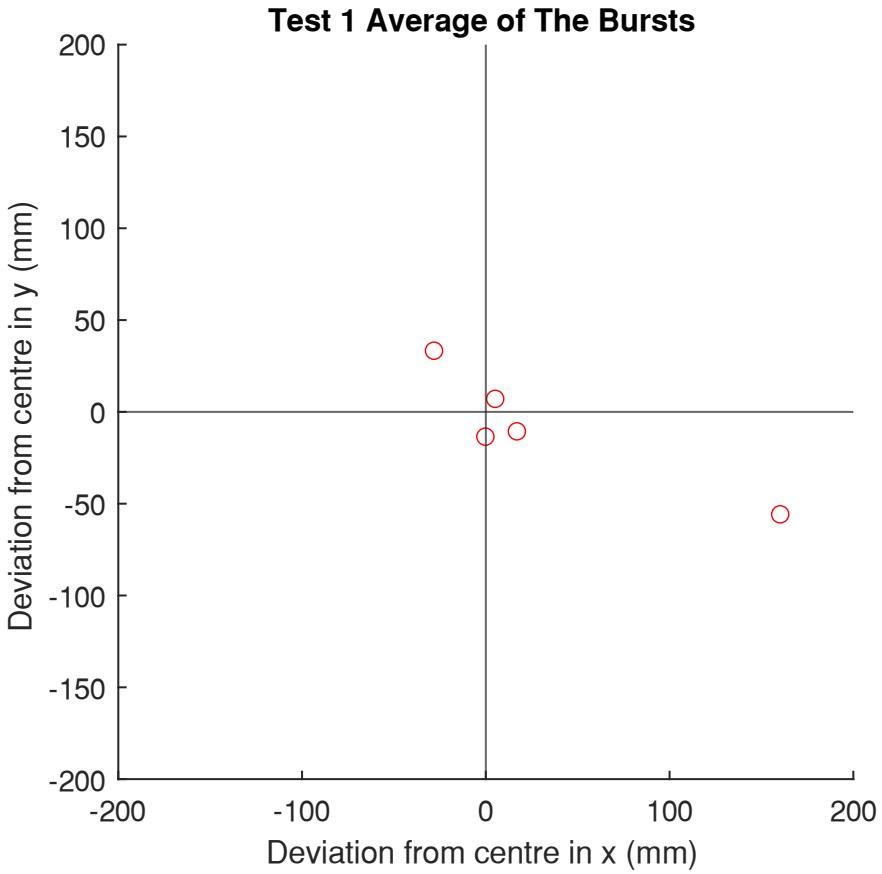
[26] Wikipedia, “Airsoft gun,” [Online]. Available: https://en.wikipedia.org/wiki/Airsoft_gun?fbclid=IwAR1sgfjWzKqpHRNkMJa2uvaZEs1zrj5bgmbhFHwXQTIRA9fKNZ2UOuf03_k#:~:text=These%20guns%20often%20attain%20muzzle,and%201500%20rounds%20per%20minute. [Accessed 10 May 2023].

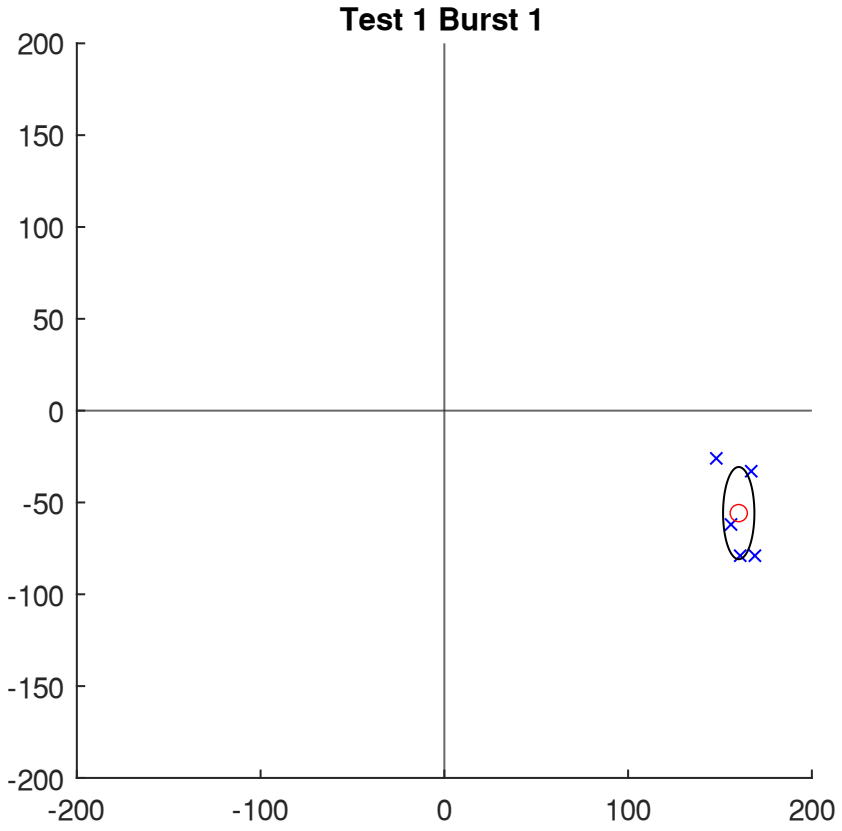
9 Appendix A

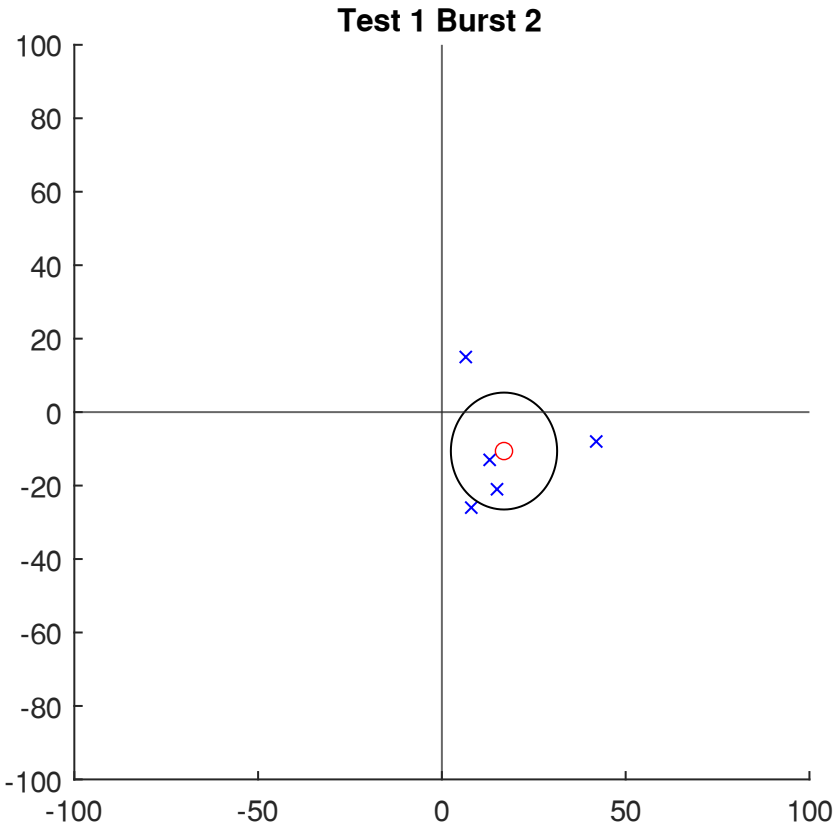
9.1 Accuracy Evaluation Target

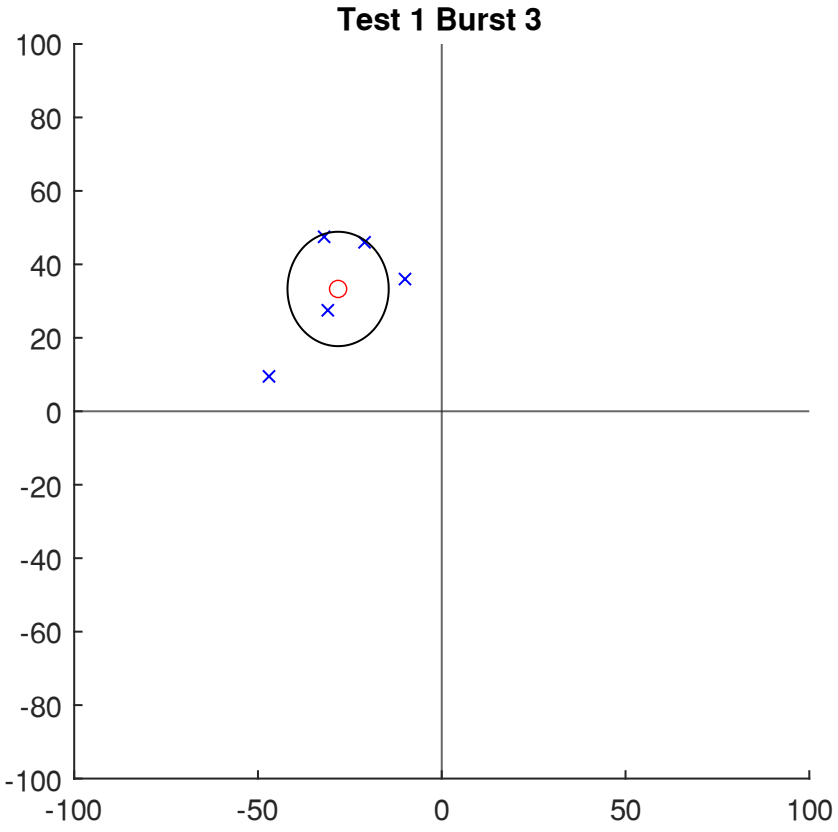


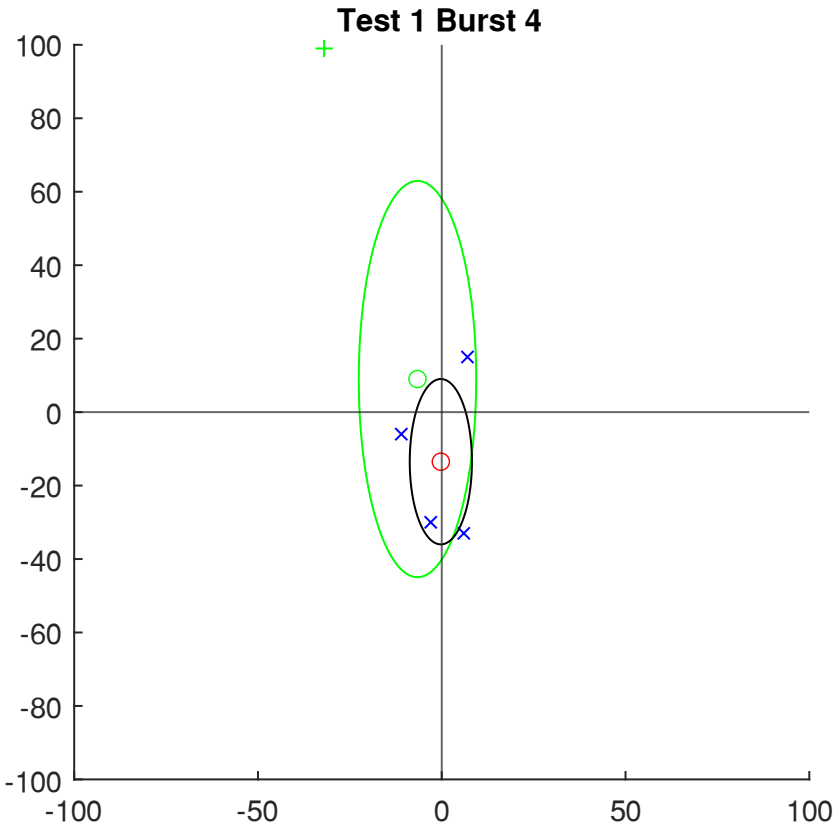
9.2 Firing Test 1

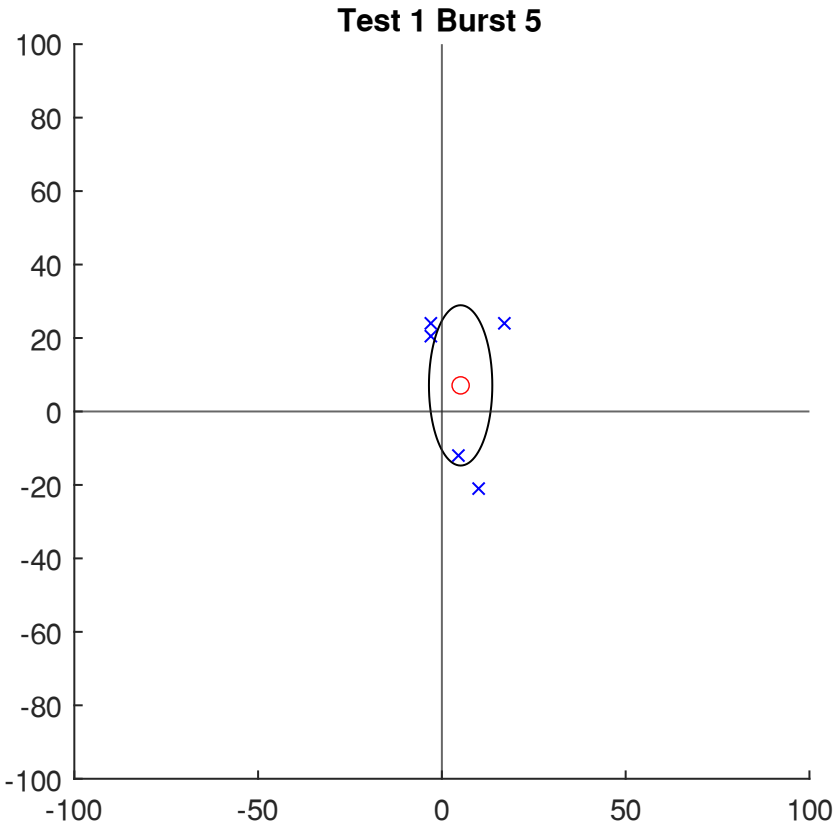




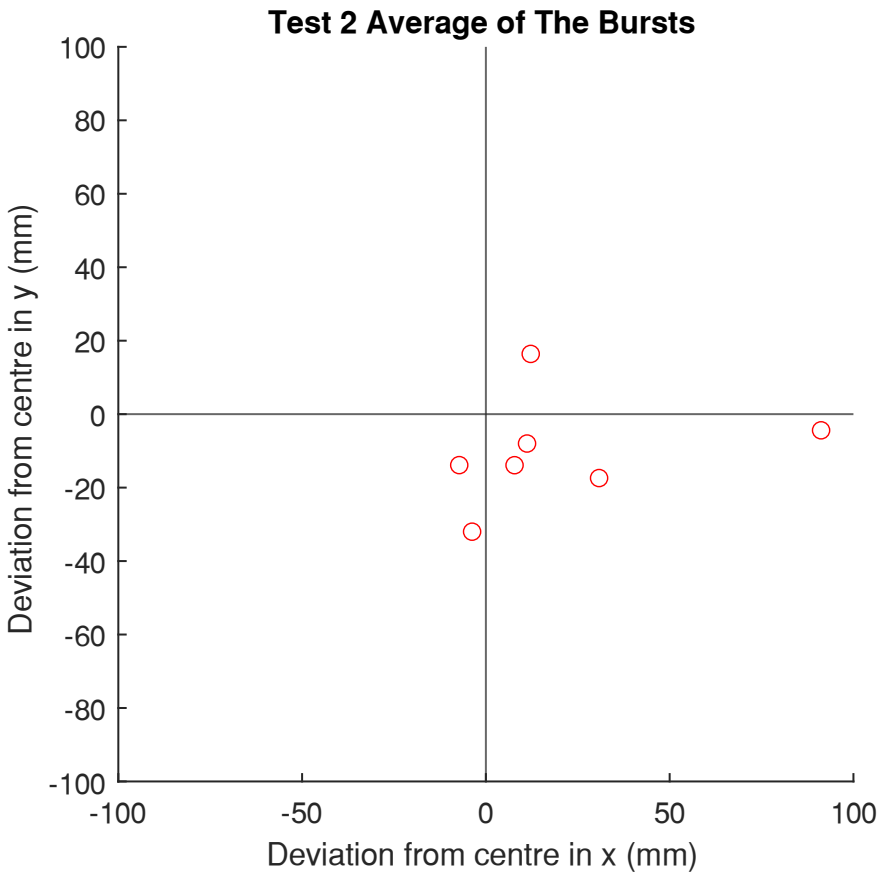

Test 1 Burst 2
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

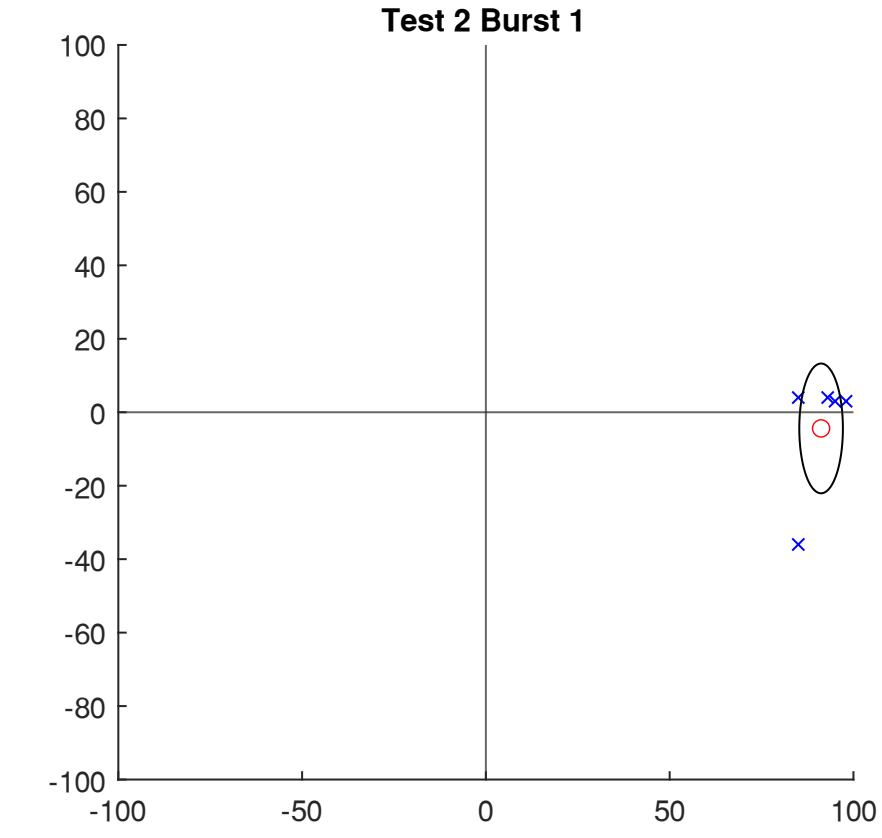

Test 1 Burst 3
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

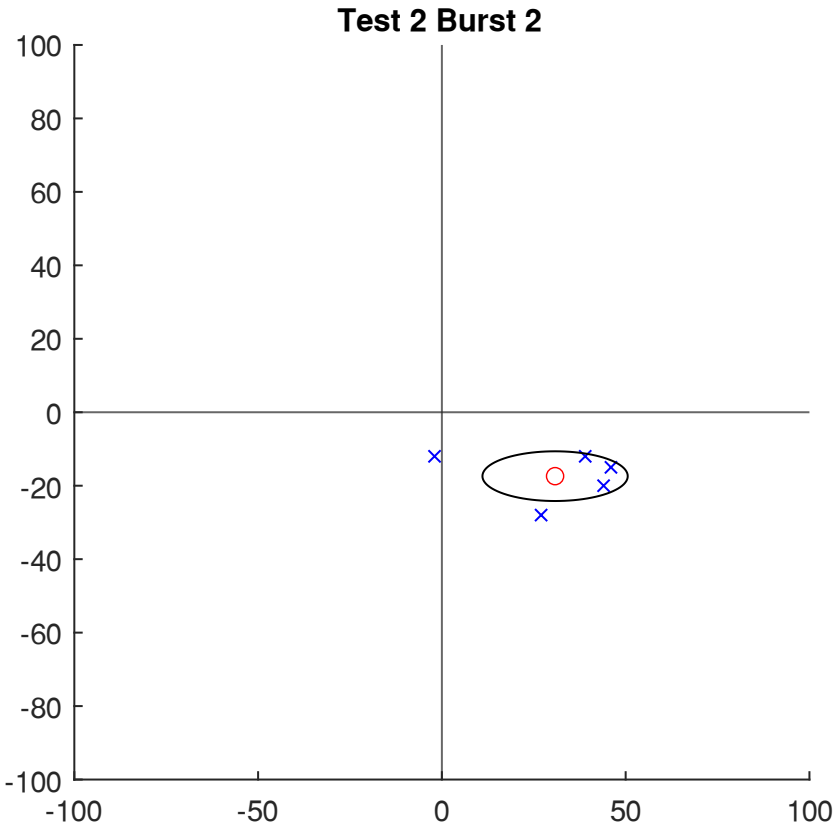

Test 1 Burst 4
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

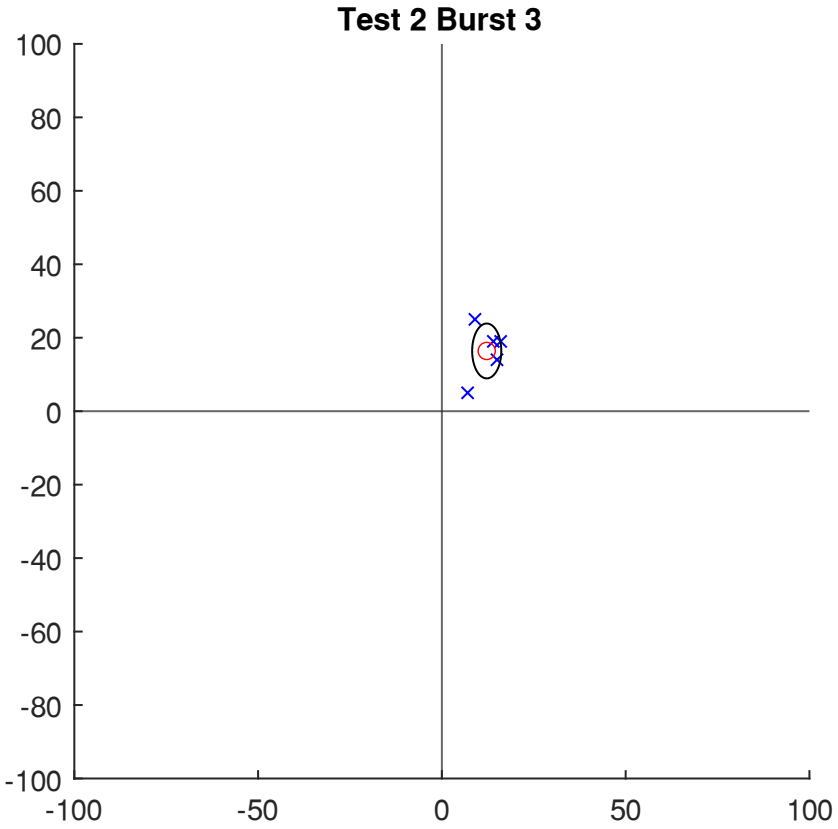

Test 1 Burst 5
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

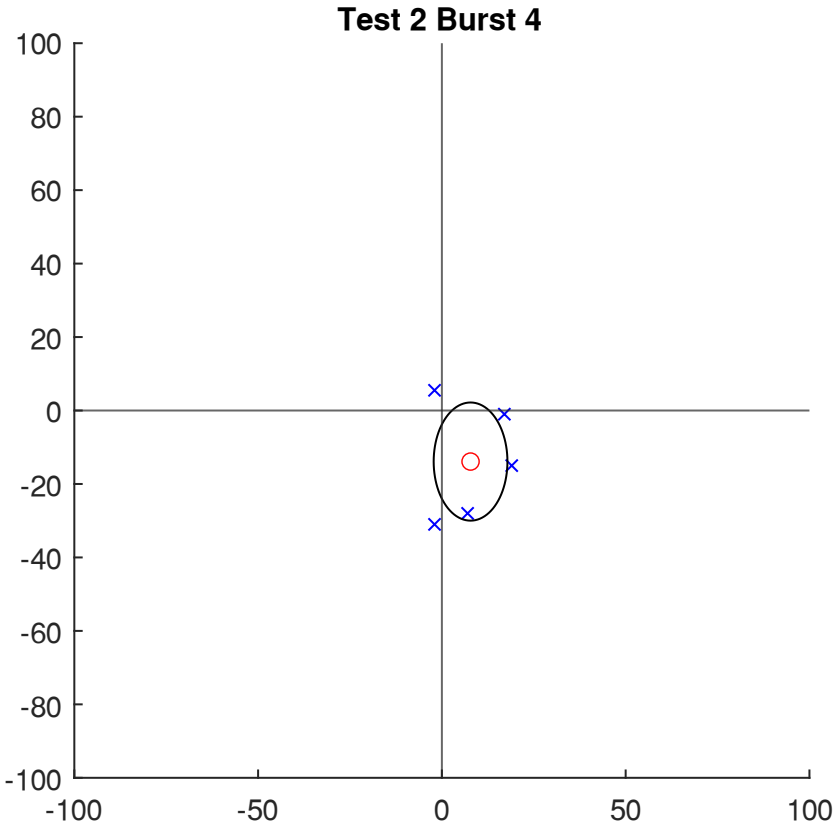
9.3 Firing test 2

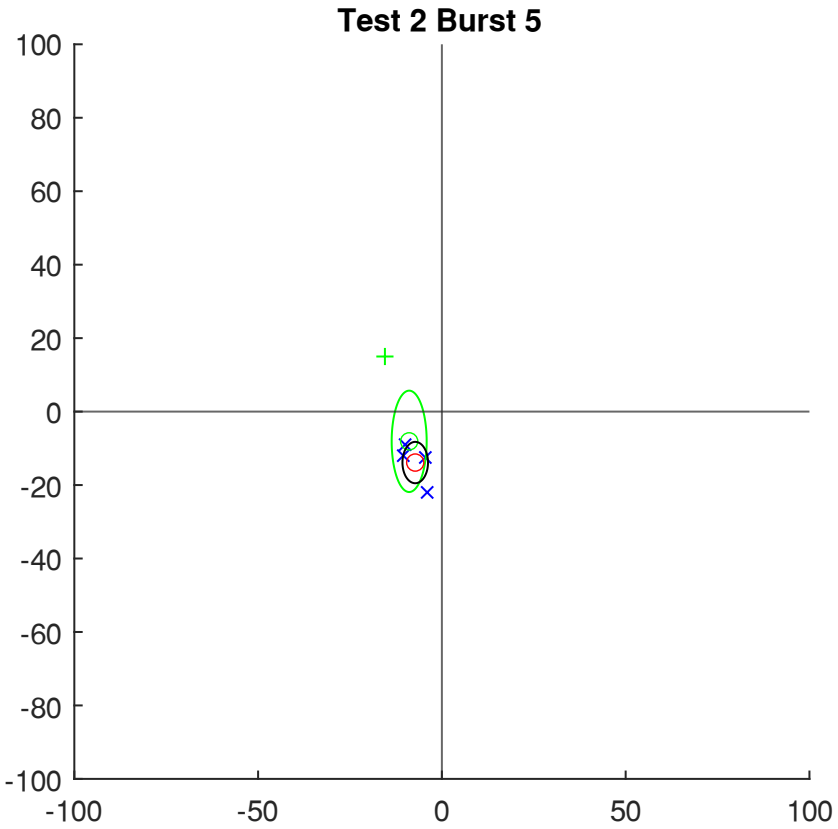


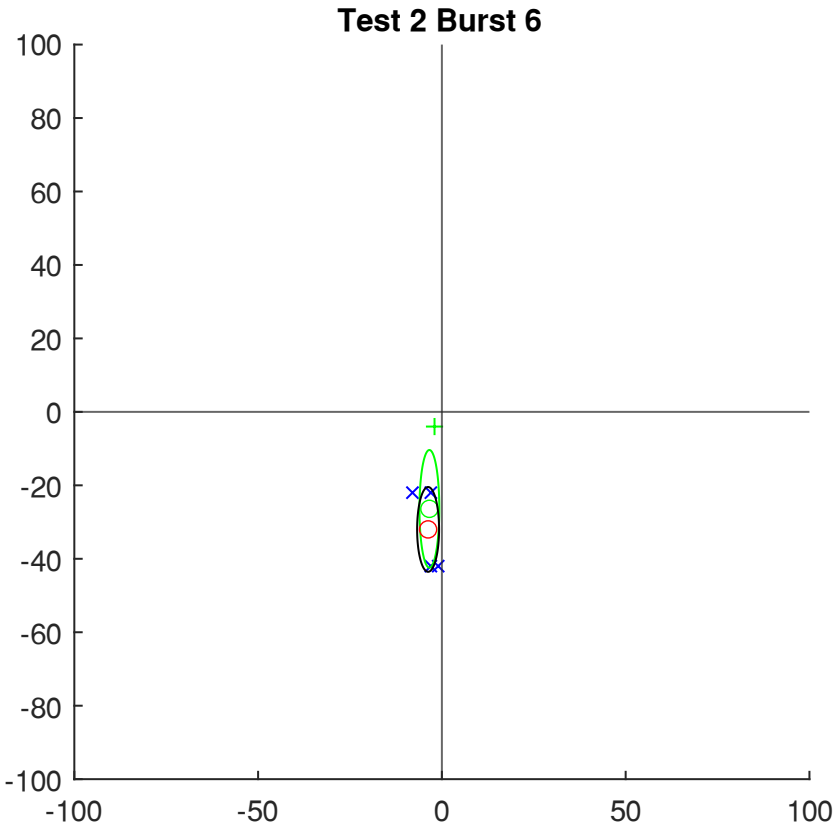


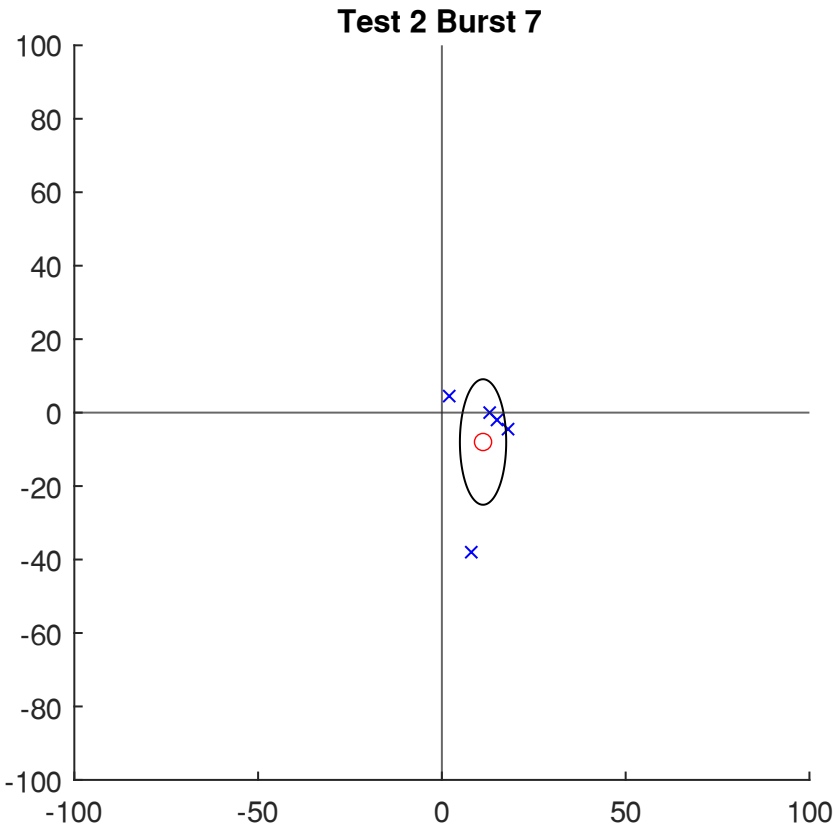

Test 2 Burst 2
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100


Test 2 Burst 3
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100


Test 2 Burst 4
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100


Test 2 Burst 5
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

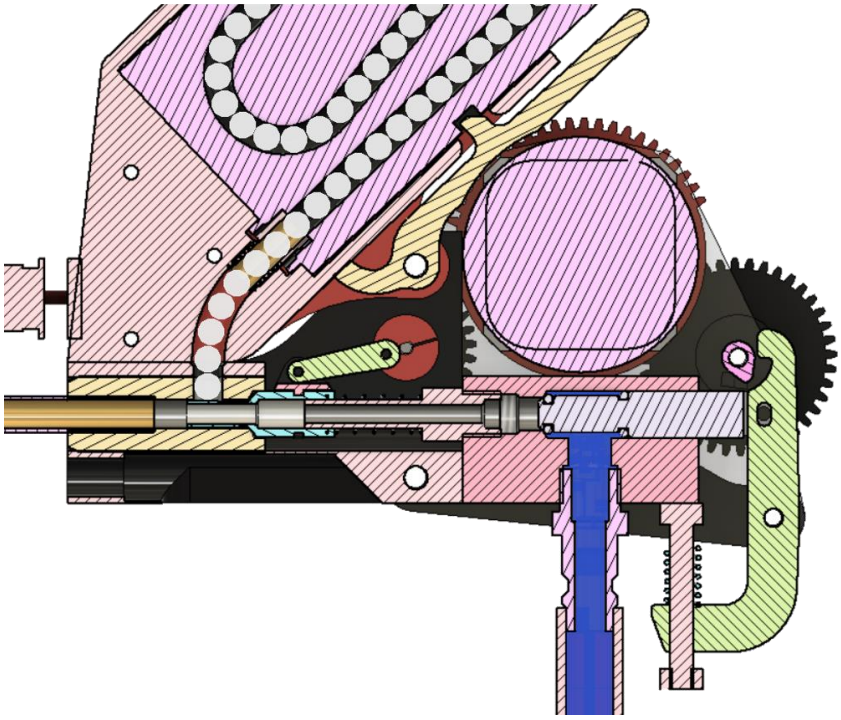

Test 2 Burst 6
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100


Test 2 Burst 7
100
80
60
40
20
0
-20
-40
-60
-80
-100
-100
-50
0
50
100

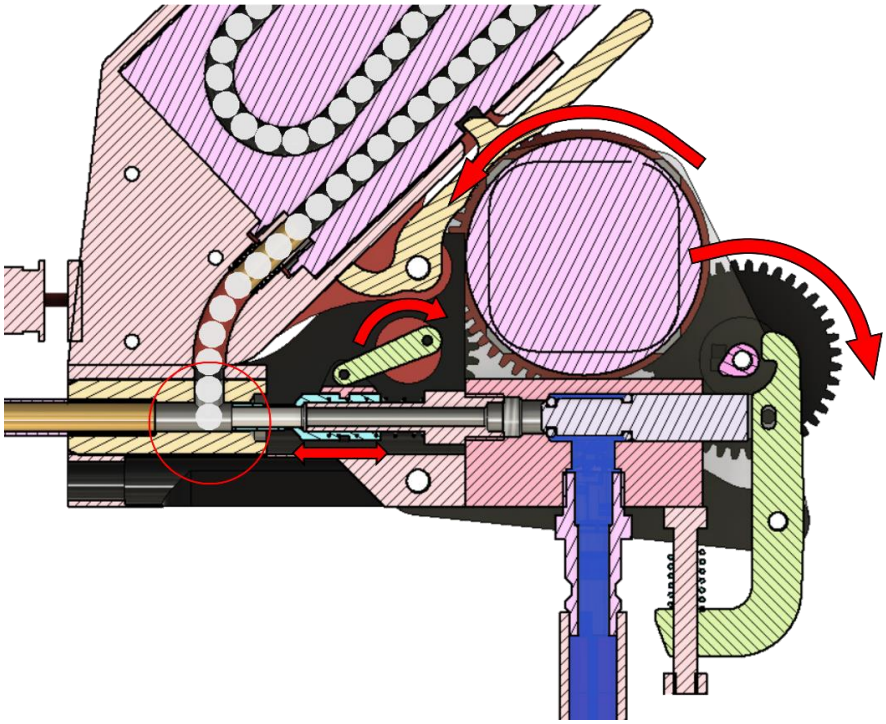
10 Appendix B

10.1 Firing Mechanism Cycle

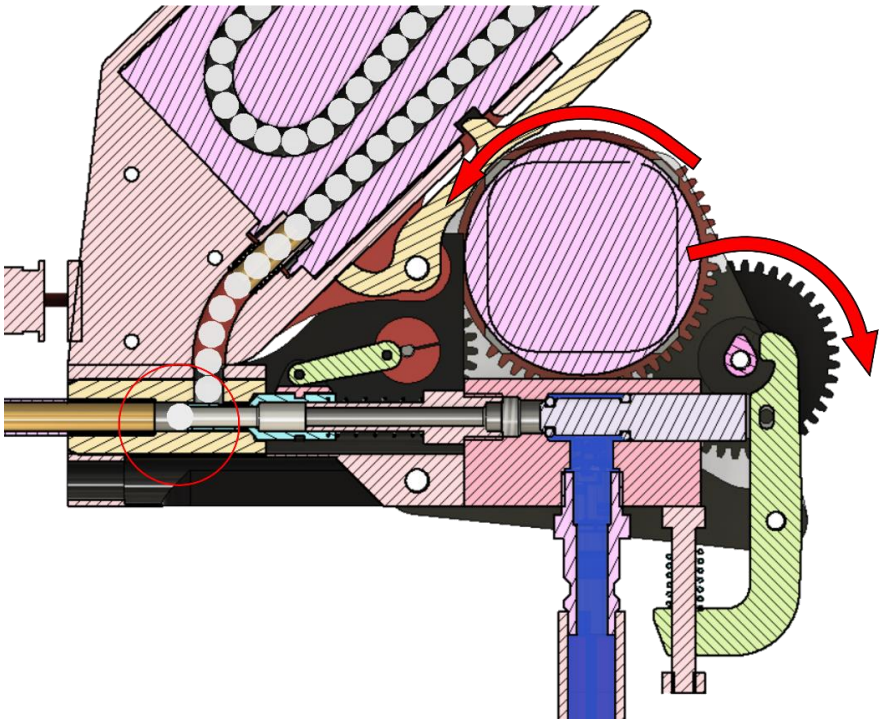
Neutral



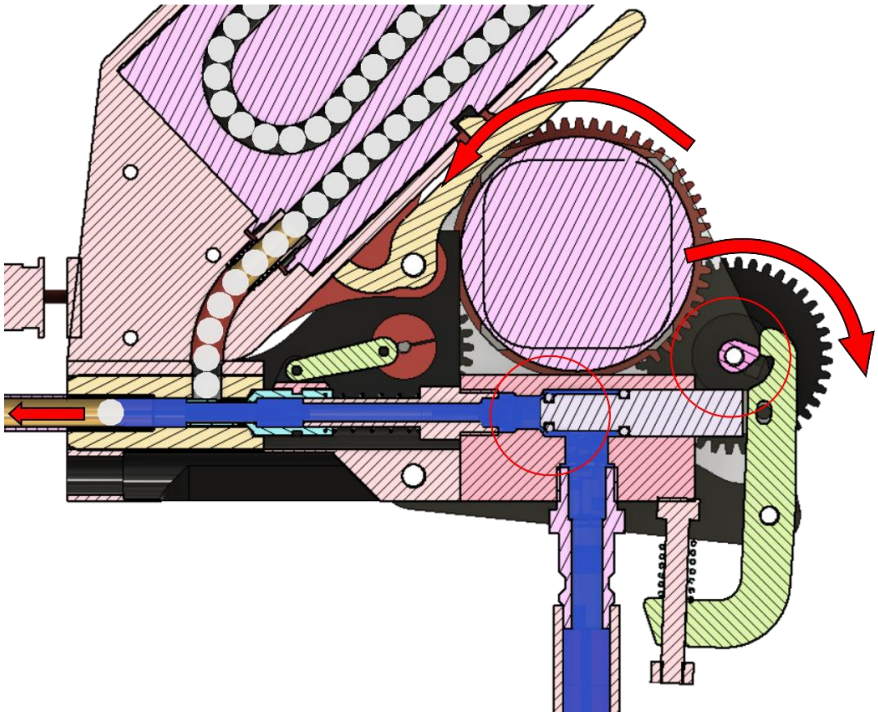
Loading



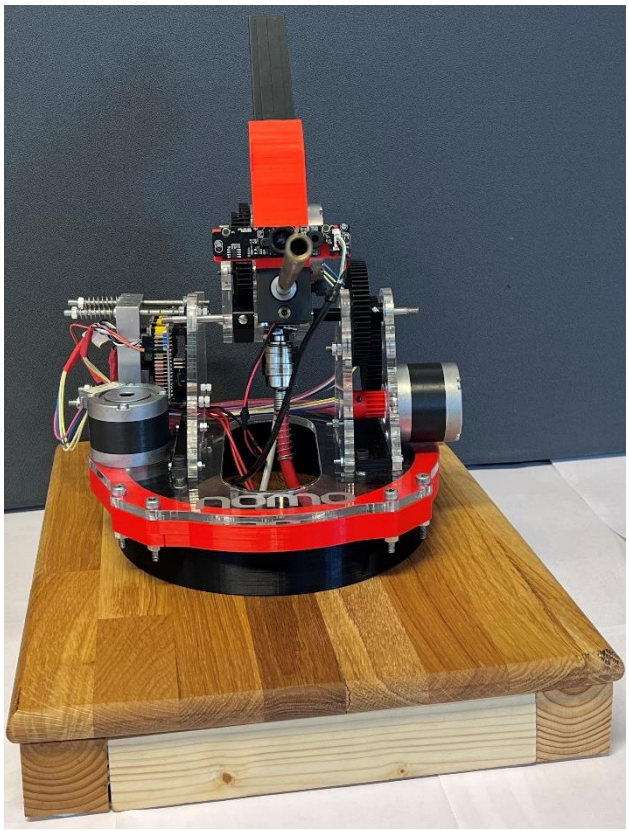
Loaded

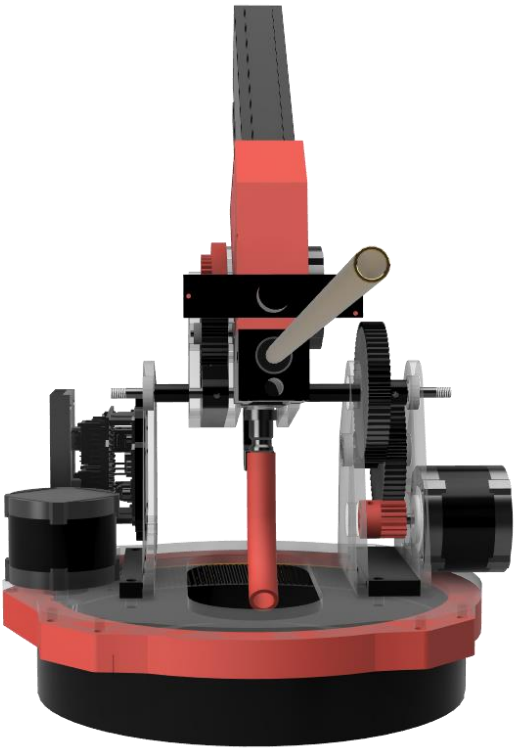


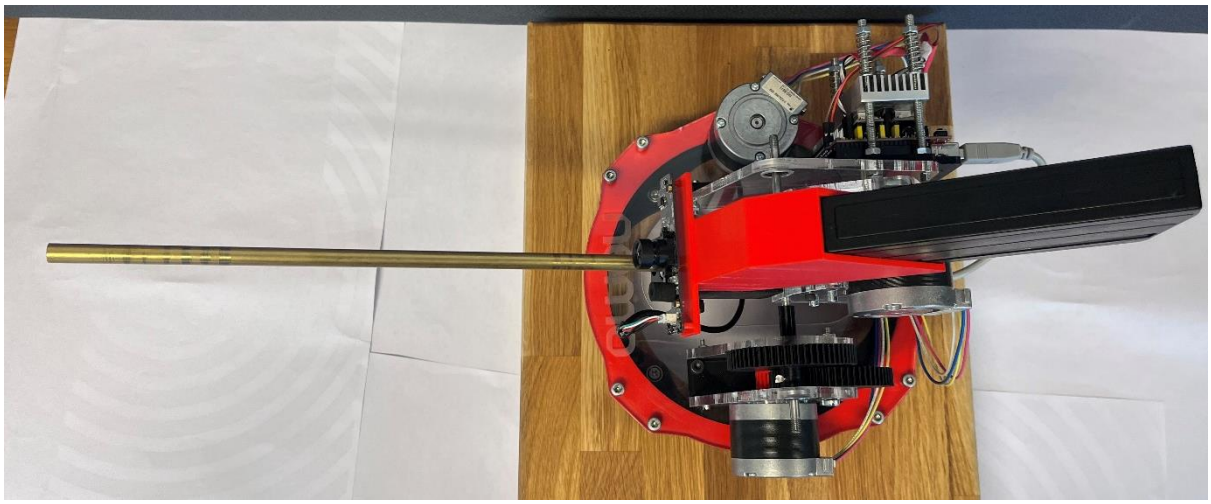
Firing

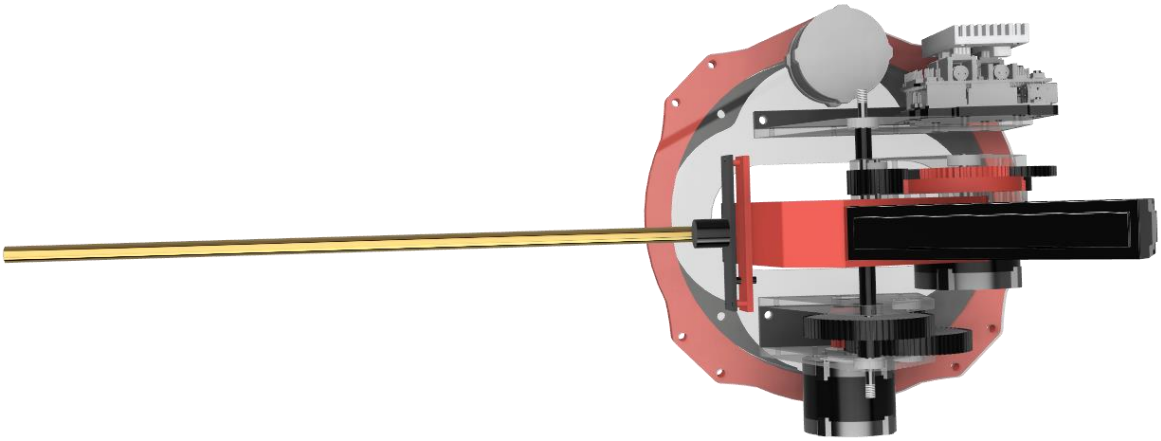


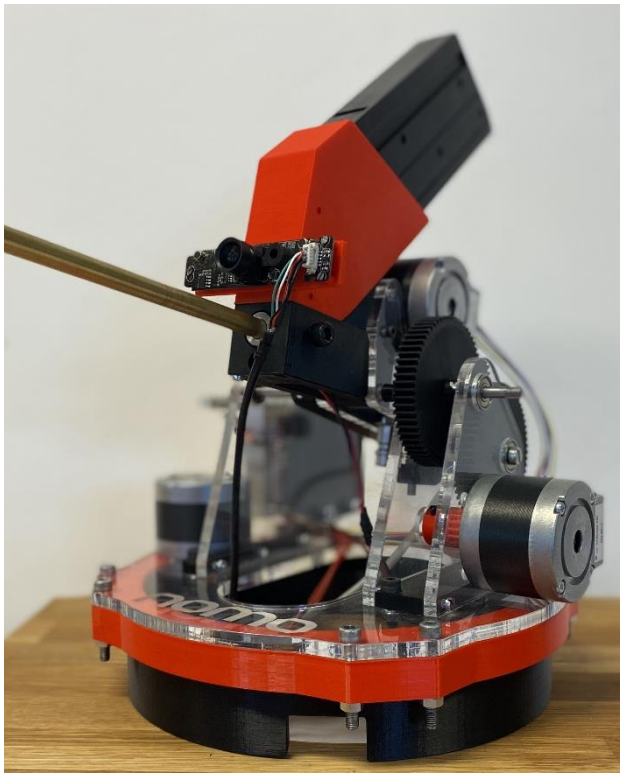
10.2 Views of the Completed Turret

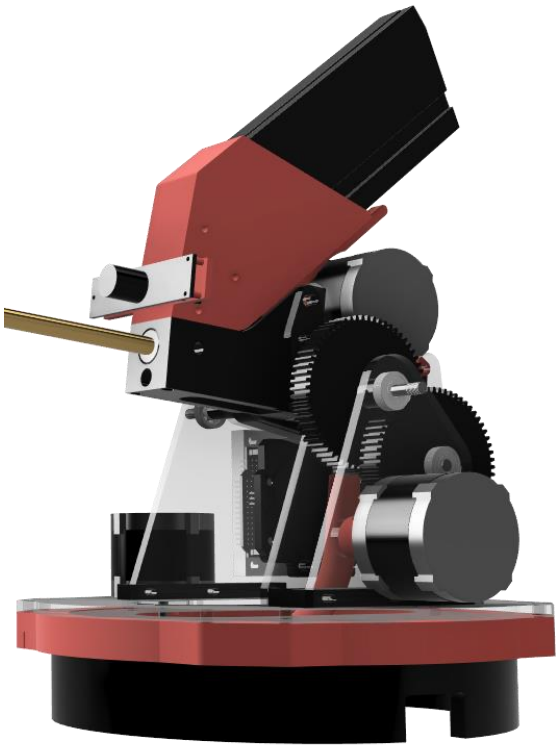


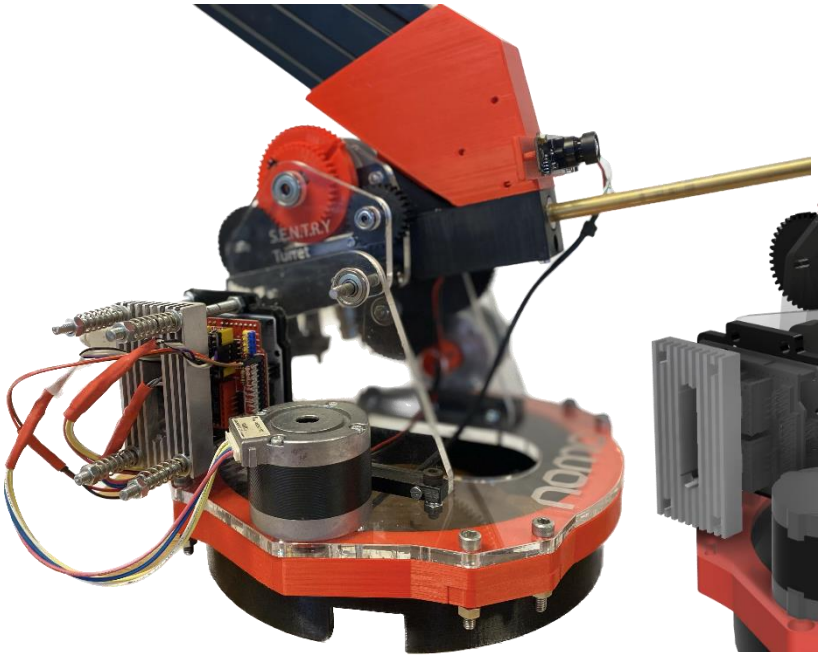


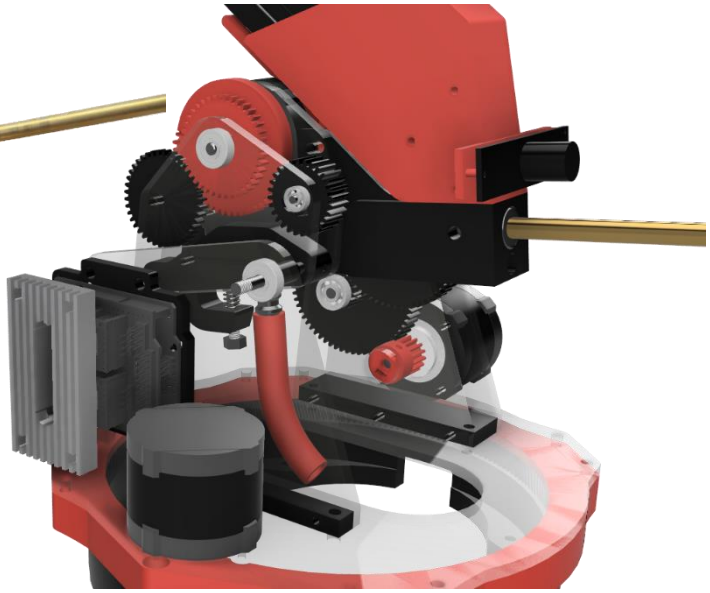


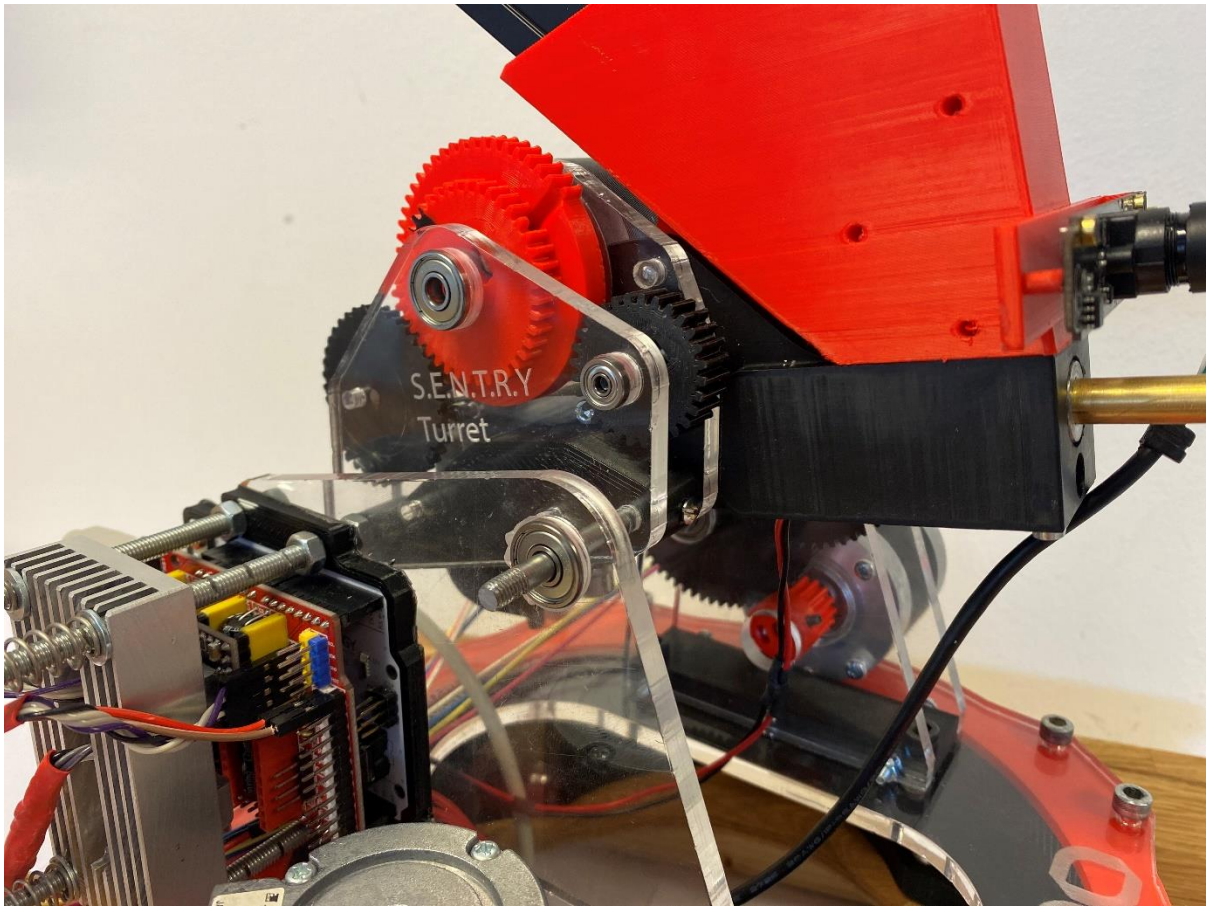


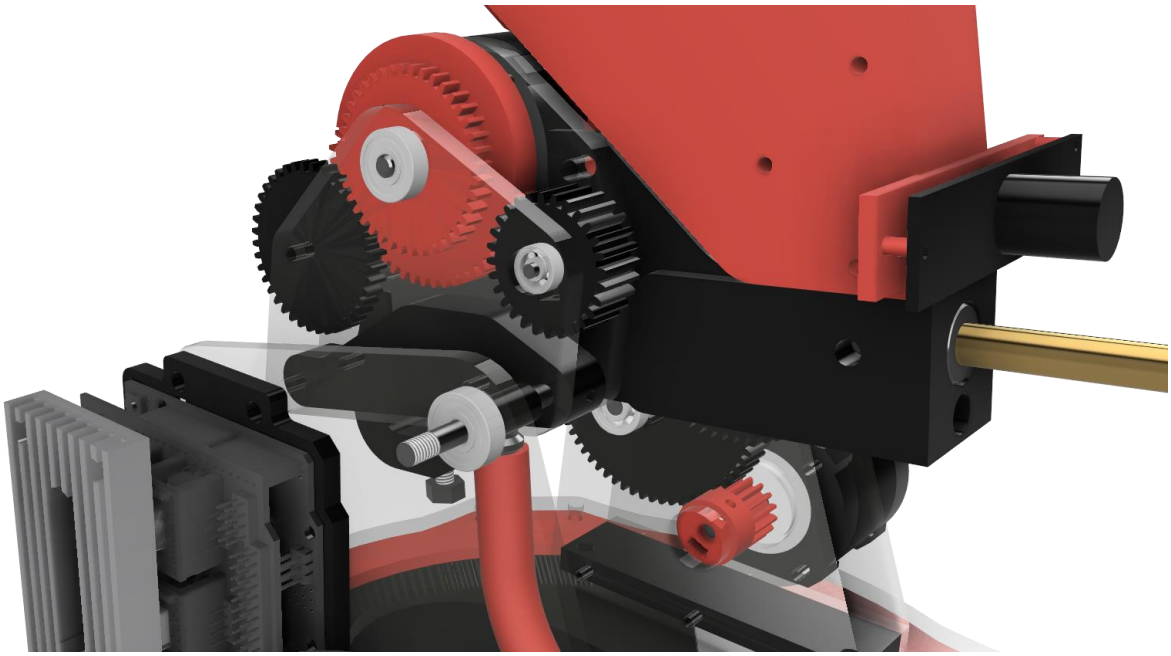


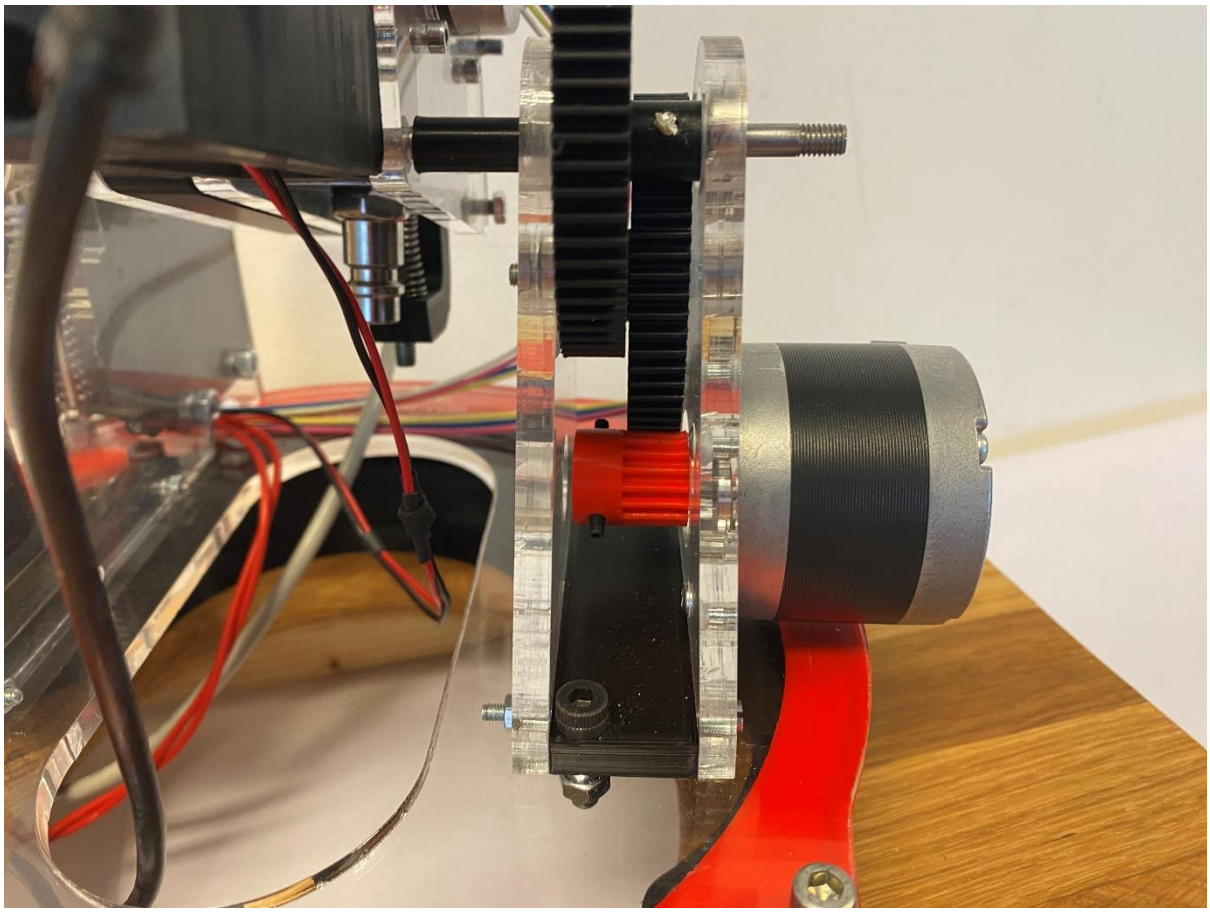


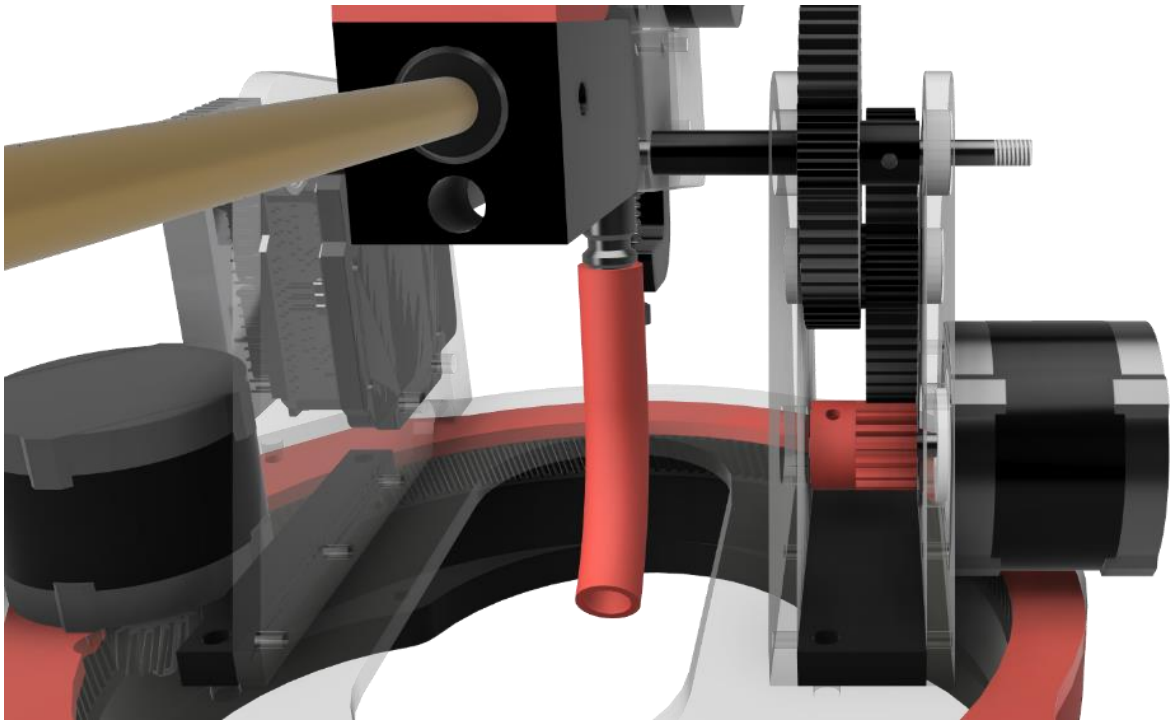

SENTRY
Turret




S.E.N.T.R.Y
Turret







11 Appendix C

11.1 Python program for turret main controls

```
#Python Code for Main Turret Controls
#Built as a Bachelor's Degree Project at KTH
#Authors:  Jacob Holst, Carl Bermhed
#Date:     2023-06-01
#Course:   MF133X Degree Project in Mechantronics, First Cycle

import tkinter as tk
import time
from Turret import Turret
import cv2
from PIL import Image, ImageTk
import colorsys
import numpy as np
import CI_turret as CI

window=tk.Tk()
window.geometry("1920x1080")

frame_a=tk.Frame(master=window,width=190,height=720)
frame_b=tk.Frame(master=window,width=150,height=720)
frame_c=tk.Frame(master=window,width=960,height=720) #creating tkinter
frames within the window

color_canvas=tk.Canvas(frame_b,width=50,height=50) #creating window and
canvas for colour id



cap = cv2.VideoCapture(1, apiPreference=cv2.CAP_ANY,
params=[cv2.CAP_PROP_FRAME_WIDTH, 960,cv2.CAP_PROP_FRAME_HEIGHT, 720])
if not cap.isOpened(): #opens camera, if now camera is attached it opens
internal camera
    cap = cv2.VideoCapture(0,
apiPreference=cv2.CAP_ANY,params=[cv2.CAP_PROP_FRAME_WIDTH, 960,
cv2.CAP_PROP_FRAME_HEIGHT, 720])

ret, capture = cap.read()
h,w,d=capture.shape
turret=Turret(w,h) #creates turret object from turret class and takes width
and height of screen

face_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+
"haarcascade_frontalface_default.xml") #imports haarcascade for facial
recognition

pitchvalue=tk.IntVar(frame_a)
yawvalue=tk.IntVar(frame_a)
delayvalue=tk.DoubleVar(frame_a)
deadzonevalue=tk.IntVar(frame_a)
autoaim=tk.StringVar(frame_a)
autoaim.set("off") #creates variables for controlling and tunign turret
during operation
```

```
def find_target(auto_aim):

    ret, capture = cap.read() #reads the camera and stores picture in
"capture"

    x_to_center=0
    y_to_center=0 #distance from center of screen to target, ramians  0 if
auto aim is turnend of
    d0=100000
    d=0 #hittar närmsta mål til center(ansikte)


    if capture is None:
        print('Error: Could not load image')
        #if camera disconnects


    if auto_aim=="face":

        gray = cv2.cvtColor(capture, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(gray, 1.3, 6) #converts
capture to grayscale and applies haarcascade to find faces

        for (x,y,w,h) in faces:
            cv2.rectangle(capture,(x,y),(x+w,y+h),(0,255,0),5) #draws
rectangle on found face
            x_center=capture.shape[1] // 2 - (x + w // 2)+turret.yaw_offset
            y_center=capture.shape[0] // 2 - (y + h //
2)+turret.pitch_offset
            d=np.sqrt(x_center*x_center+y_center*y_center) #determins the
distance from center of screen to target for x and y as well as total
distance
            if d<d0:
                x_to_center = capture.shape[1] // 2 - (x + w //
2)+turret.yaw_offset
                y_to_center = capture.shape[0] // 2 - (y + h //
2)+turret.pitch_offset
                d0=d #chooses the face closest to the center of the screen
        turret.datastring=turret.datastring+str(d)+":"

    elif auto_aim=="color":
        define_colors() #defines colours from sliders in colour tuning ui

x_to_center,y_to_center,capture=CI.dynamic(turret.rgb,turret.hue,turret.hue
Span,turret.satmin,turret.satmax,turret.valmin,turret.valmax,capture,turret
.yaw_offset,turret.pitch_offset)

        #colour identification is completed in CI_turret module

    cv2.circle(capture, (int(turret.yaw_offset+turret.width / 2),
int(turret.pitch_offset+turret.height / 2)), radius=10,
color=(0,0,255),thickness=3)
    #cv2.circle(capture,(int(turret.width / 2), int(turret.height /
2)),radius =turret.deadzone,color=(255,0,0),thickness=2)
```

```
    opencv_image = cv2.cvtColor(capture, cv2.COLOR_BGR2RGBA)
    captured_image = Image.fromarray(opencv_image)
    photo_image = ImageTk.PhotoImage(image=captured_image)
    video_pic.photo_image = photo_image
    video_pic.configure(image=photo_image)
    #draws useful information on the screen and converts capture to image
for tkinter

    return -x_to_center,y_to_center


def define_colors():

    #function takes values from colour tuning sliders and converts to HSV
format
    #also draws in colour ui screen and stores information on turret
obeject

    turret.hue = hue_slider.get()

    turret.hueSpan =hueSpan_slider.get()

    turret.satmax=satmax_slider.get()
    turret.satmin=satmin_slider.get()

    if turret.satmin>turret.satmax:
        satmin_slider.set(turret.satmax)

    turret.valmax=valmax_slider.get()
    turret.valmin=valmin_slider.get()

    if turret.valmin>turret.valmax:
        valmin_slider.set(turret.valmax)

    turret.sat=turret.satmax
    turret.val=turret.valmax


turret.rgb=colorsys.hsv_to_rgb(turret.hue/359,turret.sat/99,turret.val/99)

    r=int(turret.rgb[0]*255)
    g=int(turret.rgb[1]*255)
    b=int(turret.rgb[2]*255)
    turret.rgb=[r,g,b]
    color_canvas.itemconfig(rect, fill=f"#{r:02x}{g:02x}{b:02x}")



def run_camera():
    auto_aim=autoaim.get()
    x_center,y_center=find_target(auto_aim)
    d = np.sqrt(x_center * x_center + y_center * y_center) #finds targets
and displays picture

    if auto_aim=="color" or auto_aim=="face":
        if d>turret.deadzone:                    #if auto aim is turned on
and target is outside of the deadzon the turret will move
```

```
            x, y = turret.screentodegree(x_center, y_center)
            y, x = turret.degreetostep(y, x)
            y, x = turret.limit_check(y + turret.pitch, x + turret.yaw)
#converts screen coordinates to motor steps and checks if the turret has
nnot reched its edge
            pitchvalue.set(y)
            yawvalue.set(x)
            go_turret() #next step in moving turret
```

```
def clear_window():

    # removes any text from user input

    pitch.delete(0, 'end')
    yaw.delete(0, 'end')

def update_window(pitch,yaw):

    # updates the currnt position of motors desplayed to the user

    pitch_position_text.config(text=("Pitch = " + str(pitch)))
    yaw_position_text.config(text=("Yaw = " + str(yaw)))
    if turret.arduino_status:
        arduino_status.config(text="Arduino connected")
    else:
        arduino_status.config(text="Arduino not connected")

def go_turret():
    pitchint=turret.pitch
    yawint=turret.yaw


    try:
        pitchint = int(str(pitchvalue.get()))
    except(ValueError):
        pass
    try:
        yawint = int(str(yawvalue.get()))
    except(ValueError):
        pass

    #looks for values on the text input

    turret.move_turret(pitchint, yawint) #moves turret
    update_window(turret.pitch, turret.yaw)
    clear_window()



def fire_turret():
```

```
    #calls on turret firing method
    turret.fire_turret()

def toggle_laser():
    #calls on laser toggle method
    turret.fire_laser()

def arduino_connect():
    #attempts to connect to arduino via usb in turret class and updates
window
    turret.connect_ard()
    update_window(turret.pitch,turret.yaw)


def shift_turret(ud,rl):

    #moves turret small step from arrow input ud=up/down rl=right/left

    pitchint=turret.pitch+100*ud
    yawint=turret.yaw+100*rl

    turret.move_turret(pitchint,yawint)
    update_window(pitchint, yawint)
    clear_window()

def set_home():

    #sets the current position of the turret to 0,0 and calls for set home
method to do the same on arduino


    turret.set_home()
    update_window(turret.pitch, turret.yaw)
    clear_window()

def go_home():

    #moves turret to 0,0 position

    turret.move_turret(0,0)
    update_window(turret.pitch, turret.yaw)
    clear_window()

def auto_aim():

    #toggles what aiming method is used and updates window

    if autoaim.get()=="off":
        autoaim.set("face")
    elif autoaim.get()=="face":
        autoaim.set("color")
    elif autoaim.get()=="color":
        autoaim.set("off")
    autoaim_status_text.config(text=("Auto-Aim = " + str(autoaim.get())))

def set_delay():
```

```
    #sets delay constant given user input

    try:
        delay=delayvalue.get()
        turret.delay_constant = delay
        delay_status.config(text=("Delay constant = " +
str(turret.delay_constant)))
    except:
        pass

    clear_window()

def set_deadzone():

    #sets deadzone given user input

    try:
        deadzone=deadzonevalue.get()
    except:
        deadzone=turret.deadzone
    turret.deadzone=deadzone
    delay_status.config(text=("deadzone constant = "+str(turret.deadzone)))
    clear_window()

def grab_color(x,y,image): #image is in bgr

    #gets the avarage color of an area detirmined by "size" for more
accurate color id

    size = 1
    start_x, end_x = x - size // 2, x + size // 2
    start_y, end_y = y - size // 2, y + size // 2

    sum_r, sum_g, sum_b, sum_a = 0, 0, 0, 0

    for i in range(start_x, end_x + 1):
        for j in range(start_y, end_y + 1):
            color_rgba = image.getpixel((i, j))
            sum_r += color_rgba[2]
            sum_g += color_rgba[1]
            sum_b += color_rgba[0]

    num_pixels = size ** 2
    avg_r = sum_r // num_pixels
    avg_g = sum_g // num_pixels
    avg_b = sum_b // num_pixels

    avg_color = (avg_r, avg_g, avg_b)
    return avg_color


def handle_mousepress(event):
    x=event.x
    y=event.y

    opencv_image=capture
    captured_image = Image.fromarray(opencv_image)
```

```
    color_rgba=grab_color(x,y,captured_image)
    color_rgb = color_rgba[:3]
    turret.rgb=color_rgb #captures colour of pixels acording to mouse press

    color_hsv =
colorsys.rgb_to_hsv(color_rgb[0]/255,color_rgb[1]/255,color_rgb[2]/255)
    color_hsv=[color_hsv[0]*359,color_hsv[1]*99,color_hsv[2]*99]
    color_hsv=[round(num) for num in color_hsv] #converts colours to hsv
format

    hue_slider.set(color_hsv[0])#sets colour tuning slider acording to
captured hsv value




    x=x-(turret.width/2)
    y=-y+(turret.height/2) #converts click location origin from upper left
to middle of screen

    x,y=turret.screentodegree(x,y)

    y,x=turret.degreetostep(y,x)
    y,x=turret.limit_check(y+turret.pitch,x+turret.yaw) #converts clicked
location to motor steps and limit checks

    pitchvalue.set(y)
    yawvalue.set(x)

    go_turret() #attempts to move turret to clicked on location

def handle_quit():

    #properly closes and releases all windows and cameras

    cap.release()
    cv2.destroyAllWindows()
    window.destroy()

def handle_keypress(event):

    # handles keypresses on keybord for different actions

    key=event.keysym

    if key=="Return": #moves turret acording to user text input
        go_turret()
    elif key=="space": #toggles laser on/off
        toggle_laser()
    elif key=="Up": #moves turret pitch
        shift_turret(1,0)
    elif key=="Down":
        shift_turret(-1,0)
    elif key=="Right": #moves turret yaw
        shift_turret(0,1)
    elif key=="Left":
```

```
        shift_turret(0,-1)
    else:
        pass



#creates and places buttons, entrys, text sliders and events in the GUI


pitch_text=tk.Label(master=frame_a,text="pitch:").place(x=10,y=10)
yaw_text=tk.Label(master=frame_a,text="yaw:").place(x=10,y=40)


pitch=tk.Entry(master=frame_a,textvariable=pitchvalue)
yaw=tk.Entry(master=frame_a,textvariable=yawvalue)
delay_entry=tk.Entry(master=frame_a,textvariable=delayvalue)
pitch.place(x=50,y=10)
yaw.place(x=50,y=40)
delay_entry.place(x=50,y=70)

fire_button=tk.Button(master=frame_a,text="Fire!",command=fire_turret,width
=10).place(x=50,y=130)
go_button=tk.Button(master=frame_a,text="GO!",command=go_turret,width=10).p
lace(x=50,y=100)
set_button=tk.Button(master=frame_a,text="Set
Home",command=set_home,width=10).place(x=50,y=160)
home_button=tk.Button(master=frame_a,text="Go
Home",command=go_home,width=10).place(x=50,y=190)
autoaim_button=tk.Button(master=frame_a,text="Toggle Auto-
Aim",command=auto_aim).place(x=50,y=220)
delay_button=tk.Button(master=frame_a,text="Set
delay",command=set_delay).place(x=50,y=250)
reconnect_button=tk.Button(master=frame_a,text="Connect
arduino",command=arduino_connect).place(x=50,y=280)

sliderx=95
textx=10
starty=330
startysslide=starty-20
intervaly=40

hue_text=tk.Label(master=frame_a,text="HUE:").place(x=textx,y=starty)
hue_slider=tk.Scale(master=frame_a,from_=0,to=359,orient=tk.HORIZONTAL)
hue_slider.place(x=sliderx, y=startysslide)

hueSpan_text=tk.Label(master=frame_a,text="HUESpan +/-
:").place(x=textx,y=starty+intervaly)
hueSpan_slider=tk.Scale(master=frame_a,from_=0,to=359,orient=tk.HORIZONTAL)
hueSpan_slider.place(x=sliderx, y=startysslide+intervaly)
hueSpan_slider.set(30)

satmax_text=tk.Label(master=frame_a,text="Sat
upper:").place(x=textx,y=starty+intervaly*2)
satmax_slider=tk.Scale(master=frame_a,from_=0,to=99,orient=tk.HORIZONTAL)
satmax_slider.place(x=sliderx, y=startysslide+intervaly*2)
satmax_slider.set(99)
```

```
satmin_text=tk.Label(master=frame_a,text="Sat
lower:").place(x=textx,y=starty+intervaly*3)
satmin_slider=tk.Scale(master=frame_a,from_=0,to=99,orient=tk.HORIZONTAL)
satmin_slider.place(x=sliderx, y=startysslide+intervaly*3)
satmin_slider.set(40)


valmax_text=tk.Label(master=frame_a,text="Valspan
upper:").place(x=textx,y=starty+intervaly*4)
valmax_slider=tk.Scale(master=frame_a,from_=0,to=99,orient=tk.HORIZONTAL)
valmax_slider.place(x=sliderx,y=startysslide+intervaly*4)
valmax_slider.set(99)

valmin_text=tk.Label(master=frame_a,text="Valspan
lower:").place(x=textx,y=starty+intervaly*5)
valmin_slider=tk.Scale(master=frame_a,from_=0,to=99,orient=tk.HORIZONTAL)
valmin_slider.place(x=sliderx,y=startysslide+intervaly*5)
valmin_slider.set(15)

rect = color_canvas.create_rectangle(0, 0, 50, 50, fill="red")
color_canvas.place(x=10,y=starty)



pitch_position_text=tk.Label(master=frame_b,text="Pitch = 0")
yaw_position_text=tk.Label(master=frame_b,text="Yaw = 0")
autoaim_status_text=tk.Label(master=frame_b,text="Auto-Aim = off")
delay_status=tk.Label(master=frame_b,text="Delay constant = 0.2")
arduino_status=tk.Label(master=frame_b,text="Arduino not connected")
pitch_position_text.place(x=10,y=10)
yaw_position_text.place(x=10,y=30)
autoaim_status_text.place(x=10,y=50)
delay_status.place(x=10,y=70)
arduino_status.place(x=10,y=100)

video_pic=tk.Label(master=frame_c)
video_pic.pack()

frame_a.grid(row=0,column=0,sticky="N")
frame_b.grid(row=0,column=1,sticky="N")
frame_c.grid(row=0,column=2,rowspan=2,sticky="NE")

window.bind("<Key>", handle_keypress)
video_pic.bind("<Button-1>",handle_mousepress)
window.protocol("WM_DELETE_WINDOW", handle_quit)

#creates and places buttons, entrys, text sliders and events in the GUI

while True:

    #runs the main program and updates windows

    run_camera()
    window.update()
```

11.2 Python program for Turret Class

```
#Python Code for Turret Class
#Built as a Bachelor's Degree Project at KTH
#Authors:  Jacob Holst, Carl Bermhed
#Date:     2023-06-01
#Course:   MF133X Degree Project in Mechantronics, First Cycle

import serial
import time



class Turret: #main class for turret, includes important parameters and
methods
    def __init__(self,width,height):
        self.connect_ard() #connects arduino


        self.start_time = time.time() #timer for delay calculations
        self.limitcheck=True #True if turret should limit its movements

        self.pitch=0
        self.yaw=0 #curretn position
        self.pitch_speed=2000
        self.yaw_speed=2000 #speed of motors in steps per second
        self.pitch_ratio=1/16
        self.yaw_ratio=1/12 #gear ratio for pitch and yaw axis
        self.pitch_microstep=1/4
        self.yaw_microstep=1/4 #microstepping for pitch and yaw axis

        self.width=width
        self.height=height #size of camera picture

        self.delay_constant=0.2
        self.delay=0
        self.deadzone=8 #size of deadzone
        self.yaw_offset=0# 17 #19
        self.pitch_offset=0# 1 #negativ!!! #6

        self.rgb=[255,0,0]
        self.hue=90
        self.huemax=0
        self.huemin=0
        self.sat=255
        self.satmax=0
        self.satmin=0
        self.val=255
        self.valmax=0
        self.valmin=0 #current targeted colour

        self.camera_vy_pitch=32.5
        self.camera_vy_yaw=42.5 #camera FOV

        self.max_pitch_degree=45
        self.min_pitch_degree=-45
        self.max_yaw_degree=75
```

```
        self.min_yaw_degree=-self.max_yaw_degree #allowed movement of
turret


self.max_yaw_step=200*self.max_yaw_degree/360/self.yaw_ratio/self.yaw_micro
step
        self.min_yaw_step=-self.max_yaw_step

self.max_pitch_step=200*self.max_pitch_degree/360/self.pitch_ratio/self.pit
ch_microstep

self.min_pitch_step=200*self.min_pitch_degree/360/self.pitch_ratio/self.pit
ch_microstep #allowed movement of turret in motor steps

        self.datastring="start" #data for use in testing

    def connect_ard(self):
        #connects the computer to the arduino in the com6 usb port
        try:
            self.ard = serial.Serial(port="COM6", baudrate=14400,
timeout=.1)
            time.sleep(1)
            self.arduino_status = True
        except:
            self.arduino_status=False
            self.ard=None
            print("no arduino found on com6")

    def write_ard(self,message):

        #final step in writing to arduino
        if self.arduino_status:
            message = ("<" + message + ">")
            self.ard.write(bytes(message, "utf-8"))
        else:
            print("arduino not found")

    def move_turret(self,pitch,yaw):

        #final check befor writing to arduino

        pitch,yaw=self.limit_check(pitch,yaw)
        message=str("0,"+str(pitch)+","+str(yaw))

        if (time.time()-self.start_time)>(self.delay):

            #insted of blockin with time.delay() program checks that enough
time has suprased
            #befor writing to arduino again

            self.calc_delay(pitch, yaw)
            self.start_time=time.time() #calculates delay befor next move
can be taken

            self.pitch=pitch
            self.yaw=yaw
            self.write_ard(message) #writes final message to arduino
```

```
    def fire_laser(self):

        #writes a message to move to its current position and sends a
special command of 1 to toggle the laser

        message = str("1," + str(self.pitch) + "," + str(self.yaw))
        self.write_ard(message)

    def fire_turret(self):

        #writes a message to move to its current position and sends a
special command of 3 to fire the turret once

        message=str("3," + str(self.pitch) + "," + str(self.yaw))
        self.write_ard(message)

    def set_home(self):

        #writes a message to move to its current position and sends a
special command of 2 to set teh current motor position to 0

        message = str("2," + str(self.pitch) + "," + str(self.yaw))
        self.write_ard(message)

        self.pitch=0
        self.yaw=0

    def print_movement(self,pitch,yaw):

        #prints the attempted move in degrees for testing parameters

        pitchdeg = self.pitch - pitch
        pitchdeg=pitchdeg/(200/self.pitch_microstep/self.pitch_ratio)*360
        yawdeg=self.yaw-yaw
        yawdeg=yawdeg/(200/self.yaw_microstep/self.yaw_ratio)*360
        print("pitch moves "+str(pitchdeg)+" degres")
        print("yaw moves "+str(yawdeg)+" degres")

self.datastring+=str(self.deadzone)+"_"+str(pitchdeg)+"_"+str(yawdeg)+":"
        print(self.datastring)

    def split_colors(self,hsv,delta=40):

        #splits colors to a upper and lower hue (not currently used)

        upper_hue=self.color_check(hsv[0]+delta)
        lower_hue=self.color_check(hsv[0]-delta)
        upper=[upper_hue,255,255]
        lower=[lower_hue,0,0]
        return upper,lower

    def color_check(self,hue):

        #moves the hue arround the 255 corner (not currently used)
```

```
        if hue>255:
            hue=hue#-255
        elif hue<0:
            hue=hue#+255
        return hue

    def screentodegree(self,x,y):

        #converts screen coordinates to screen angle calculated from the
screen FOV

        yawdeg=(self.camera_vy_yaw*x)/(self.width/2)
        pitchdeg=(self.camera_vy_pitch*y)/(self.height/2)

        return yawdeg,pitchdeg

    def degreetostep(self,pitchdeg,yawdeg):

        #converts degree movement to number of motor steps


pitch_step=int(round(((self.max_pitch_step*pitchdeg)/self.max_pitch_degree)
))

yaw_step=int(round(((self.max_yaw_step*yawdeg)/self.max_yaw_degree)))


        pitch_step,yaw_step=self.limit_check(pitch_step,yaw_step)

        return pitch_step,yaw_step

    def limit_check(self,pitch,yaw):

        #checks if the attempted move is beyond its allowed movement and if
so sets that movement to the edge

        pitch=pitch
        yaw=yaw
        if self.limitcheck:
            if pitch>self.max_pitch_step:
                pitch=self.max_pitch_step

            elif pitch<self.min_pitch_step:
                pitch=self.min_pitch_step

            if yaw > self.max_yaw_step:
                yaw = self.max_yaw_step

            elif yaw < self.min_yaw_step:
                yaw = self.min_yaw_step
        return int(pitch),int(yaw)

    def calc_delay(self,pitch,yaw):

        #calculates the delay based on how long the movement should take
and adds a constant to the delay
```

```
        pitch_d=abs(pitch-self.pitch)
        yaw_d=abs(yaw-self.yaw)

        pitch_t=pitch_d/self.pitch_speed
        yaw_t=yaw_d/self.yaw_speed

        delay=pitch_t
        if yaw_t>delay:
            delay=yaw_t #the delay should be equal to to longest movement
necesary

        delay=delay+self.delay_constant
        self.delay=delay
```

11.3 Python Code for Colour identification

```
#Python Code for Colour Identification and processing
#Built as a Bachelor's Degree Project at KTH
#Authors:  Jacob Holst, Carl Bermhed
#Date:     2023-06-01
#Course:   MF133X Degree Project in Mechantronics, First Cycle

import cv2  # RGB 0-255 för alla HSV Hue 0-179, Sat och val 0-255
import numpy as np

def select_largest_contour(contours):
    # initialize largest area and contour
    largest_area = 400  # Arbiträrt valt minstavärde
    largest_contour = None

    # loop over the contours
    for contour in contours:
        area = cv2.contourArea(contour)

        if area > largest_area:
            largest_area = area
            largest_contour = contour

    return largest_contour




def pixel2Array(pixel):
    # format pixel=np.uint8([[[Hue,Sat,Val]]])
    array = np.array([pixel[0][0][0], pixel[0][0][1], pixel[0][0][2]])
    return array

def minmax(value,min,max):
    if value<=min:
        value=min
    if value>=max:
        value=max
    return value

def RGBA2HSVspan(RGBA, hue, hueSpan, satMin, satMax, valMin, valMax):
    # RGBAformat: [RED,GREEN,BLUE,255] ex.[221,0,48,255] Span in degrees:

    searchColourBGR = np.uint8([[[RGBA[2], RGBA[1], RGBA[0]]]])  # Färgen
som sökes
    searchColourHSV = cv2.cvtColor(searchColourBGR, cv2.COLOR_BGR2HSV)

    lowerHue=round(minmax((hue-hueSpan)//2,0,179)) #Konvertera till 179
grader
    upperHue=round(minmax((hue+hueSpan)//2,0,179))

    lowerSat=round(minmax((satMin/99)*255,0,255))
    upperSat=round(minmax((satMax/99)*255,0,255))

    lowerVal=round(minmax((valMin/99)*255,0,255))
    upperVal=round(minmax((valMax/99)*255,0,255))
```

```
    lowerHSV = np.uint8([[[lowerHue, lowerSat, lowerVal]]])
    upperHSV = np.uint8([[[upperHue, upperSat, upperVal]]])

    lowerBGR = cv2.cvtColor(lowerHSV, cv2.COLOR_HSV2BGR)  # Färg för
display
    upperBGR = cv2.cvtColor(upperHSV, cv2.COLOR_HSV2BGR)  # Färg för
display

    lowerArray = pixel2Array(lowerHSV)
    upperArray = pixel2Array(upperHSV)

    lowerListBGR = pixel2Array(lowerBGR).tolist()
    upperListBGR = pixel2Array(upperBGR).tolist()
    searchColourListBGR = pixel2Array(searchColourBGR).tolist()

    return lowerArray, upperArray, lowerListBGR, upperListBGR,
searchColourListBGR


def dynamic(RGB, hue, hueSpan, satMin, satMax, valMin,
valMax,video,yaw_offset,pitch_offset):

    width=video.shape[1]
    height=video.shape[0]
    x_center = width//2
    y_center = height//2

    colourLower, colourUpper, lowerBGR, upperBGR, colourBGR =
RGBA2HSVspan(RGB, hue, hueSpan, satMin, satMax, valMin, valMax)


    HSVimg = cv2.cvtColor(video, cv2.COLOR_BGR2HSV)  # Converting BGR image
to HSV format
    mask = cv2.inRange(HSVimg, colourLower, colourUpper)  # Masking the
image to find our color

    mask_contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL,
                                                cv2.CHAIN_APPROX_SIMPLE)  #
Finding contours in mask image



    # Finding position of all contours
    if len(mask_contours) != 0:
        largest_contour = select_largest_contour(mask_contours)


        x, y, w, h = cv2.boundingRect(largest_contour)
        # Centrum av det största målet.
        centre_x = ((2 * x + w) // 2)+yaw_offset
        centre_y = ((2 * y + h) // 2)+pitch_offset

        cv2.rectangle(video, (x, y), (x + w, y + h), colourBGR, 3)  #
drawing rectangle
```

```
        cv2.circle(video, (centre_x, centre_y), 10, (0, 0, 255), 1)  #
drawing rectangle
        x_center=width//2-(x+w//2)+yaw_offset
        y_center=height//2-(y+h//2)+pitch_offset

    cv2.imshow("bild", mask)




    return x_center,y_center,video
```

11.4 Arduino Program for turret movement

```
// Arduino code for sentry turret
// Built as a Bachelor's Degree Project at KTH
// Authors:  Jacob Holst, Carl Bermhed
// Date:     2023-06-01
// Course:   MF133X Degree Project in Mechantronics, First Cycle

#include <AccelStepper.h>
#include <MultiStepper.h>

AccelStepper pitch(1, 3, 6);
AccelStepper yaw(1, 2, 5);
AccelStepper turret(1, 4, 7); //initialize stepper controlls with pins 3,6
2,5 4,7 according to cnc shield specs

int laserPin = 11;

bool laser=false; //laser pin and status

MultiStepper steppersControl; //steppers to controll with multistepper

long gotoposition[3]; //position for mutlistepper to move

const byte numChars = 32; //maximum number of characters to revice
char receivedChars[numChars];
char tempChars[numChars];  // temporary array for use when parsing

int yawFromPC = 0;
int pitchFromPC = 0;
int commandFromPC = 0;//variables to hold the parsed data
bool newData = false;

//============

void setup() {
  Serial.begin(14400);
  pinMode(laserPin, OUTPUT);

  pitch.setMaxSpeed(2000);
  pitch.setAcceleration(500);
  pitch.setCurrentPosition(0);

  yaw.setMaxSpeed(2000);
  yaw.setAcceleration(500);
  yaw.setCurrentPosition(0);

  turret.setMaxSpeed(5000);
  turret.setAcceleration(400);
  turret.setCurrentPosition(0); //defines parameters for motors

  steppersControl.addStepper(pitch);
  steppersControl.addStepper(yaw);
  steppersControl.addStepper(turret);
}

//============
```

```
void loop() {

  recvWithStartEndMarkers(); //attempt to recive new data
  if (newData == true) {
    strcpy(tempChars, receivedChars);
    parseData(); //turn message into useful data

    gotoposition[0] = -pitchFromPC;  //gear ratio forces movement other
direction
    gotoposition[1] = yawFromPC; //gives position for attempted move
    gotoposition[2]=0;
    steppersControl.moveTo(gotoposition); //calculates pulse timings for
stepper movement

    steppersControl.runSpeedToPosition(); //moves steppers to calculatde
location

    if (commandFromPC == 1) { //toggles laser on and off

      if (not laser){
        analogWrite(laserPin, 125); //125 for limiting voltage

      }else if (laser){
        analogWrite(laserPin, 0);
      }
      laser=not laser;
      digitalWrite(LED_BUILTIN,HIGH);
      delay(1000);
      digitalWrite(LED_BUILTIN,LOW);
    }
    else if (commandFromPC==2){ //sets current position to 0,0 for set home
command in python on computer
      pitch.setCurrentPosition(0);
      yaw.setCurrentPosition(0);
    }
    else if (commandFromPC==3){ //rotates firing stepper one rotation to
fire one projectile
      gotoposition[0] = pitch.currentPosition();
      gotoposition[1] = yaw.currentPosition();
      gotoposition[2]=800;
      steppersControl.moveTo(gotoposition);
      steppersControl.runSpeedToPosition();
      turret.setCurrentPosition(0); //
    }


    newData = false; //when data us used it is cosiderd old
  }
}

//============

void recvWithStartEndMarkers() {
  static boolean recvInProgress = false;
  static byte ndx = 0;
  char startMarker = '<';
```

```
  char endMarker = '>';
  char rc;

  while (Serial.available() > 0 && newData == false) {
    rc = Serial.read(); // reads next character in buffer

    if (recvInProgress == true) {
      if (rc != endMarker) {
        receivedChars[ndx] = rc;
        ndx++;
        if (ndx >= numChars) {
          ndx = numChars - 1;
        }
      } else { //if charcter is an end marker (>) a complete string of data
is ready
        receivedChars[ndx] = '\0';  // terminate the string
        recvInProgress = false;
        ndx = 0;
        newData = true;
      }
    }

    else if (rc == startMarker) {
      recvInProgress = true;
    }
  }
}
//============
void parseData() {  // split the data into its parts

  char* strtokIndx;  // this is used by strtok() as an index

  strtokIndx = strtok(tempChars, ",");
  commandFromPC = atoi(strtokIndx);

  strtokIndx = strtok(NULL, ",");
  pitchFromPC = atoi(strtokIndx);

  strtokIndx = strtok(NULL, ",");
  yawFromPC = atoi(strtokIndx);  //converts everything seperated with a ,
to an int to three commands
}
```

11.5 Matlab Velocity Plot

```
d = 0.006; % diameter of the pellet (m)
A=d^2/2*pi();%area m^2
L = 0.363; % length of the barrel (m)
m=0.2/1000; % mass of the pellet (kg)
pStep=0.01
P = pStep:pStep:8; % over pressure [range] (bar)

roh=1.225;% air density (kg/m^3) at standard conditions
Cd=0.47; % drag coefficient for a sphere
dt=(0.363/340)/200 %Step in time (s)

v_exit = zeros(size(P));
for i = 1:length(P)
    p=P(i);
    x=0; %m
    v=0; %m/s
    a=0; %m/s^2

    while x<=L
        Fdrag=1/2*roh*v^2*Cd*A;
        a=(A*p*10^5-Fdrag)/m;
        v=v+a*dt;
        if v>343
            v=343;
        end
        x=x+v*dt;
    end
    v_exit(i) = v;
end
v_exit

plot(P,v_exit)
xlabel('Pressure (bar)')
ylabel('Exit Velocity (m/s)')
title('Exit Velocity vs. Pressure for a 6mm Pellet in a 363mm
Barrel')
```