Bachelor gauntlet auto-following paintball gun

Final Report

J.R. Lund
21448745

Submitted as partial fulfilment of the requirements of Project EPR402
in the Department of Electrical, Electronic and Computer Engineering
University of Pretoria

November 2024

Study leader: Dr JH van Wyk

Part 1. Preamble

This report describes the work I did in designing an autonomous computer-vision paintball gun using discrete electronics, and Python software.

Project proposal
This main report contains an unaltered copy of the approved Project Proposal (as Part 2 of the report).

All the code that I developed appears as a separate submission on the AMS.

Project history
This project does not build on any previous projects by other students. Simulation software was provided by Siemens and MATLAB. The PCB manufacturing was completed from the first principles by myself. Some of the algorithms I used were adapted from Robben [1], [2], [3], and [4]. The rest of the work reported on here, is entirely my own.

Language editing
This document has been language edited by a knowledgeable person. By submitting this document in its present form, I declare that this is the written material that I wish to be examined on.

My language editor was _____.

_____ _____

Language editor signature *Date*

Declaration

I, Jack Lund understand what plagiarism is and have carefully studied the plagiarism policy of the University. I hereby declare that all the work described in this report is my own, except where explicitly indicated otherwise. Although I may have discussed the design and investigation with my study leader, fellow students or consulted various books, articles or the internet, the design/investigative work is my own. I have mastered the design and I have made all the required calculations in my lab book (and/or they are reflected in this report) to authenticate this. I am not presenting a complete solution of someone else.

Wherever I have used information from other sources, I have given credit by proper and complete referencing of the source material so that it can be clearly discerned what is my own work and what was quoted from other sources. I acknowledge that failure to comply with the instructions regarding referencing will regarded as plagiarism. If there is any doubt about the authenticity of my work, I understand that I may have to attend an oral ancillary examination/evaluation about the work.

I certify that the Project Proposal appearing as the Introduction section of the report is a verbatim copy of the approved Project Proposal.

J.R. Lund

Date

TABLE OF CONTENTS

Part 2. Project definition: approved Project Proposal

1. Project description
2. Technical challenges in this project
3. Functional analysis
4. System requirements and specifications
5. Field conditions
6. Student tasks

Part 3. Main Report

1. Literature study 1
    1.1. Computer vision solution 1
        1.1.1. CNN 2
    1.2. PID and hardware 4
    1.3. Contribution 4

2. Approach 5
    2.1. Hardware 5
    2.2. Software 5

3. Design and implementation 7
    3.1. Design summary 7
    3.2. System block diagram 8
    3.3. Theoretical Background 11
    3.4. Software design 13
        3.4.1. Model Architecture 13
        3.4.2. Image processing 16
        3.4.3. Forward pass 18
        3.4.4. Loss function 24
        3.4.5. Filtering results 26
        3.4.6. Real-time detection and visualisation 28
        3.4.7. Utilities 30
        3.4.8. PID 33
    3.5. Hardware design 37
        3.5.1. PCB 37
        3.5.2. Turret 38
    3.6. Configuration 42
        3.6.1. Training 42
        3.6.2. Inference 44

4. Results 46
    4.1. Summary of results achieved 46
    4.2. Qualification tests 46
        Qualification test 1: Confidently distinguish the Bachelor from the 'friendly' person

Objectives of the test or experiment 46
Equipment used 46
Test setup and experimental parameters 47
Steps followed in the test or experiment 47
Results or measurements 48
Observations 50
Statistical Analysis 51
Qualification test 2: Person object detection and recognition
Objectives of the test or experiment 53
Equipment used 53
Test setup and experimental parameters 53
Steps followed in the test or experiment 53
Results or measurements 54
Observations 55
Qualification test 3: The PID controller, turret, and frame rate
Objectives of the test or experiment 56
Equipment used 56
Test setup and experimental parameters 56
Steps followed in the test or experiment 57
Results or measurements 57
Observations 58
Statistical Analysis 59
Qualification test 4: Multiple object tracking
Objectives of the test or experiment 60
Equipment used 60
Test setup and experimental parameters 60
Steps followed in the test or experiment 60
Results or measurements 61
Observations 61

5. Discussion 63
5.1. Critical evaluation of the design 63
5.1.1. Interpretation of results 63
5.1.2. Critical evaluation 64
5.1.3. Unsolved problems 65
5.1.4. Strong points of the design 65
5.1.5. Expected failure conditions 65
5.2. Considerations in the design 66
5.2.1. Ergonomics 66
5.2.2. Health and safety 66
5.2.3. Environmental impact 66
5.2.4. Social and legal impact 67
5.2.5. Ethics clearance 67

6. Conclusion 68
6.1. Summary of the work completed 68
6.2. Summary of the observations and findings 68
6.3. Contribution 68
6.4. Future work 69

7. References 70

Appendix

Datasets 72

Safety 74

LIST OF ABBREVIATIONS

PWM	Pulse-Width Modulation
I2C	Inter-Integrated Circuit
CNN	Convolutional Neural Network
YOLO	You Only Look Once
CG	Center of Gravity
PCB	Printed Circuit Board
LCD	Liquid Crystal Display
PID	Proportional – Integral – Derivative
FPS	Frames Per Second
CNN	Convolutional Neural Network
NN	Neural Network
TPR	True positive Rate
mAP	Mean Average Precision
ANN	Artificial Neural Networks
NMS	Non-Maximum Suppression
IoU	Intersection Over Union
FLOP	Floating Point Operations
FLOPS	Floating Point Operations per Second
UI	User Interface
CSV	Comma-Separated Values
ROI	Region Of Interest
CEMF	Counter-Electromotive Force
EMF	Electromotive Force
BCE	Binary Cross Entropy
MSE	Mean Squared Error
GPU	Graphics Processing Unit
GIGO	Garbage In, Garbage Out
CUDA	Compute Unified Device Architecture

Part 2. Project definition: approved Project Proposal

This section contains the problem identification in the form of the complete approved Project Proposal, unaltered from the final approved version that appears on the AMS.

For use by the Project lecturer	Approved		Revision required	

Feedback

For use by the Project module lecturer only

To be completed by the student

PROJECT PROPOSAL 2024

Project no	JHvW4	Revision no	1

Title	Surname	Initials	Student no	Study leader (title, initials, surname)
Mr	**Lund**	**JR**	21448745	Dr JH van Wyk

Project title (the title on the project concept note)

Bachelor gauntlet auto-following paintball gun

Language editor details	Language editor signature
Mr VWX Vite	
Student declaration I understand what plagiarism is and that I have to complete my project on my own.	**Study leader declaration** This is a clear and unambiguous description of what is required in this project. Approved for submission (Yes/No)
Student signature	Study leader signature and date 9 June 2024

1. Project description

What is the problem to be solved with your project? What is your project about? What does your system have to do?

Autonomous warfare is a very controversial industry but small-scale projects for preventative measures are starting to emerge. These systems fill the gaps in security where human operation becomes unreasonable. This project is a stationary paintball gun turret that has been given the task of identifying 'the bachelor', tracking and following the target, and shooting only the 'unfriendly', 'bachelor'. This project is intended to take part in the gauntlet run where one person dressed as a 'bachelor' has to run though a designated area between two rows of people while the rows of people shoot the 'bachelor', the automated gun is to replace the two rows of people. The system will also be able to be used as an autonomous security system where it classifies known people as 'friendly' and unknown as 'the bachelor'. It is purely a preventative measure, not a lethal measure. The system will use computer vision to identify people and then classify them as a 'friendly' or a 'unfriendly'/'the bachelor'. The system then uses a control system to control a 2-axis gimbal to manoeuvre the paintball gun to follow the bachelor and shoot the target with paintballs accurately. Hereafter the bachelor is defined as the person whom the system must follow and shoot.

2. Technical challenges in this project

Describe the technical challenges that are *beyond* those encountered up to the end of third year and in other final year modules.

2.1 Primary *design* challenges

Which aspects of the design of the system do you expect to be the most challenging?

This project is both software and hardware-intensive. The software requirements include, a computer vision system (which has never been done in any undergraduate modules) that needs to be designed to detect objects, classify them and track them (multi-object tracking). Out of the objects classified as humans, it then needs to identify the 'friendly' from the bachelor. This system then has to feed inputs to a separate proportional–integral–derivative (PID) control system which controls motors to manoeuvre a paintball gun to follow the target, and issue commands to shoot the 'unfriendly'/bachelor, with some degree of accuracy. On the hardware side, a chassis needs to be designed to hold the gun, trigger servo, processing platform camera setup, and gimbal design.

2.2 Primary *implementation* challenges

Which aspects of the implementation to you expect to be the most challenging?

The main implementation challenge is a result of the nature of the processing requirements needed to perform high-speed processing on a live graphic stream (live video input), to perform live object recognition and tracking, requiring a fairly powerful graphics processor, not found on a common MCUs such as an ESP32 or an STM32F4 series. One needs to make use of a processing platform designed for graphics processing. The smaller challenge is ensuring the motors used to position the gun have enough static torque to hold the gun in place while enduring the 'kick' and weight of the gun. The integration of the camera, processing platform, motors, and servos also requires hardware to be designed, e.g. H-bridges.

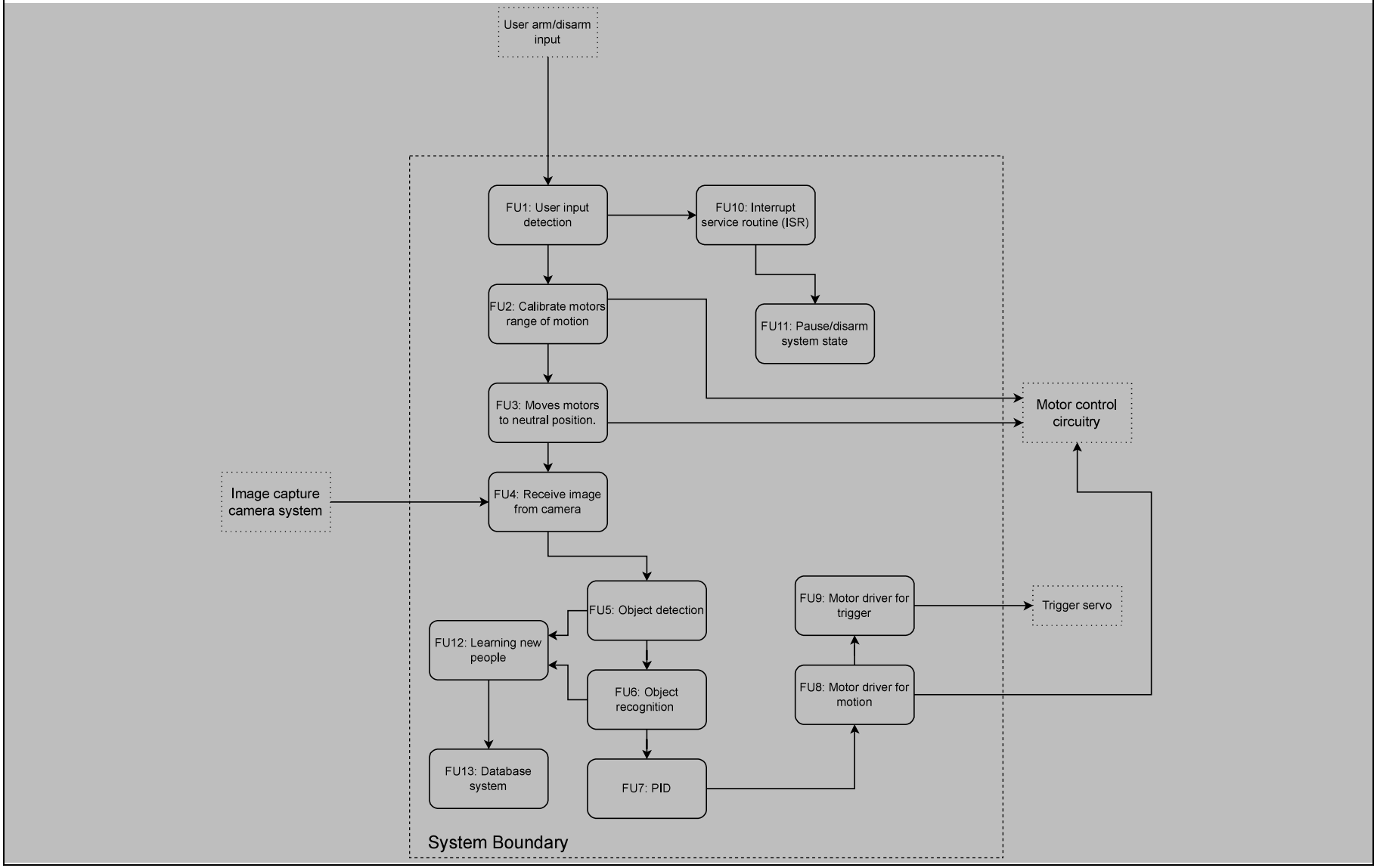
3. Functional analysis

3.1 Functional description

Describe the design in terms of system functions as shown on the functional block diagram in section 3.2. This description should be in *narrative format*. **DO NOT** use a bullet list.

The user input detection (FU1) allows the systems to detect the user input through a switch or button to arm/disarm the system. Calibrate motors range of motion (FU2) involves the calibration system for the 2-axis gun gimbal for the system to learn the limits of the motors, similar to how a 3d-printer calibrates on start-up. The system must then calculate the neutral/middle position of the motor's range of motion (FU3) and wait to receive images from the video camera system (FU4), preprocess the image and perform object detection (FU5). The processed images are then fed to the artificial intelligence (AI) system (FU6) where objects are recognized and classified/ identified. If the bachelor is detected then the PID controller for the gimbal motors (FU7) is used to manoeuvre the barrel of the gun to point at the bachelor. The system must make use of pulse width modulation (PWM) (FU8) to control the motors through motor driver circuitry. When the barrel is aligned with the predicted position of the bachelor, the PWM to control the trigger servo (FU9) on the paintball gun must be used to pull the trigger through servo driver circuitry. If at any time the system is disarmed, the system must stop instantly using the interrupt service routine (ISR) (FU10), and transition to the paused/disarmed state (FU11). The system needs to be able to add people the the 'friendly' database and therefore, learning new people (FU12) is needed to add new 'friendly' known people and store their information in a database (FU13).

3.2 Functional block diagram (this should not be a flow diagram)



4. System requirements and specifications

These are the core requirements of the system or product (the mission-critical requirements) in table format **IN ORDER OF IMPORTANCE**. Requirement 1 is the most fundamental requirement.

	Requirement 1: the fundamental functional and performance requirement of your project	**Requirement 2** (Number 2 in the order of importance)	**Requirement 3** (Number 3 in the order of importance)
1. Core mission requirements of the system or product. Focus on requirements that are **core** to solving the engineering problem. These will reflect the solution to the problem.	The system must distinguish the bachelor from 'friendly' people.	The system must be able to detect and distinguish people from other objects and animals.	The PID system must ensure the barrel of the gun's set point is always the center of the bachelor with overshoot to account for the time of flight (ToF).
2. What is the target specification (in *measurable* terms) to be met in order to achieve the requirement in 1. above?	The system must correctly identify the bachelor 75% of the time in an area of 30m (width) by 20m (depth) and should never incorrectly identify a 'friendly' person as the bachelor.	The system must identify a human among other objects with a true positive rate (TPR) of 95%.	The system must constantly update the set point of the PID to follow the 'unfriendly' target. The system must be have up to a 0.65m overshoot.
3. Motivation: Defend the specific target specification, i.e. the value that you selected. I.e., *why* will meeting the specification given in point 2 above *solve the problem*?	For a defence system, a false negative is acceptable but a false positive is not. Research shows home invasions normally involve one to three invaders, therefore, with 3 invaders, the probability at least one is detected is 98.44%.	Due to the statistical nature of the algorithms used for object recognition, the distribution of the TPR is normalised around a mean, 95% is chosen because 95% is 2 standard deviations.	The next set point for the aim must be constantly updated since the objects will be moving. The speed of the ball is 90 m/s, and the average speed of man is 3.25 m/s, so at 20 m from the gun ToF= 0.2 s. Therefore the human can move 0.65m.
4. How will you demonstrate at the examination that this requirement and specification (points 1 and 2 above) have been met? Be explicit about how you will *prove* these were met.	Multiple 'friendly' persons will pass in front of the system and the system should not identify any as the bachelor. The bachelor will then enter the systems field of view multiple times, the systems should detect the bachelor within specification.	Many objects will be placed in the camera's field of view and a handful of people, including willing invigorators, will walk past the camera 10 times, the system must detect the people 9 times. This can be run in tandem with requirement 1.	A monitor will be connected to the GPU of the controller and cross-hairs of the set point will be shown in each frame. The overshoot will be seen by the system aiming "ahead" of the moving target and will be in the simulations of the PID.
5. Your own design contribution: what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement.*	The object detection and recognition algorithm will be designed and implemented.	The algorithm used for object detection will be designed and implemented.	The PID values will be calculated and MATLAB simulations used to design and test the PID systems.
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none". Clearly specify for what tasks **library functions** will be used (if relevant to the project).	The processing platform for processing the images, and the camera for capturing the frames to be processed will be purchased.	The system will also be trained on open-source images of people in order to ensure proper training. The processing platform that will perform the graphics processing. The camera that captures the images. OpenCV only to get images from the camera.	MATLAB to simulate the PID will be used as an off the shelf program.

System requirements and specifications page 2

	Requirement 4	Requirement 5	Requirement 6
1. Core mission requirements of the system or product. Focus on requirements that are core to solving the engineering problem. These will reflect the solution to the problem.	Multiple object tracking, the system should be able to track up to 5 people at once.	Frame rate is the biggest challenge. The system must be fast enough to process frames fast enough to be able to have an effective object-tracking system.	The turret must be able to swivel across the full field of view of the camera.
2. What is the target specification (in measurable terms) to be met in order to achieve the requirement in 1. above?	The system must be able to identify 5 people in one frame.	The system must be able to keep the gun aimed at a moving 'unfriendly' at 5 frames/sec.	At a distance of 20 meters perpendicular to the camera lens' face, the camera should be able to view 30 meters in diameter and the turret must be able to place the centre of its scope in any position in that range from starting position.
3. Motivation: Defend the specific target specification, i.e. the value that you selected. I.e., *why* will meeting the specification given in point 2 above solve the problem?	In the gauntlet run the system must be able to follow and shoot the running bachelor and track the other 4 players ('friendly') on the course who are not the bachelor but are additional shooters trying to hit the bachelor or spectators.	The system has to be able to track objects motion multiple times a second in order to predict movements. The human reaction time is 200-300 ms, therefore the gun must be slightly faster (200 ms).	The system must be able to move in a large area to give the system enough time to acquire the moving target and hit it.
4. How will you demonstrate at the examination that this requirement and specification (points 1 and 2 above) have been met? Be explicit about how you will *prove* these were met.	Two invigilators or colleagues will enter the camera's field of view at once and the system should draw boxes around both people in each frame which must be shown on a monitor.	A timer and frame counter will run to give a frames per second (FPS) measurement that will be displayed. Requirement 3 shows a man can move 0.65 m in 0.2 seconds, every frame will show the current aim is no more than 0.65 m off.	20 and 30 meter diameters will be drawn. An 'unfriendly' will walk across the 30 meter line. The system will be observed move and keep the centre position of the gun barrel in line with the center of the 'unfriendly' target the whole time.
5. Your own design contribution: what are the aspects that *you will design and implement yourself* to meet the requirement in point 2? If none, *remove this requirement.*	The object detection and recognition algorithm will be designed and implemented.	The objection recognition and tracking algorithm, will be designed for efficiency and implemented.	The gimbal's chassis and movement algorithms will be designed and implemented. Drivers for the motors will be designed and implemented.
6. What are the aspects to be taken off the shelf to meet this requirement? If none, indicate "none". Clearly specify for what tasks **library functions** will be used (if relevant to the project).	OpenCV to get the images, and a camera will be taken off the shelf.	The camera will be taken off the shelf.	Sheet metal and plastic for 3D printing will be taken off the shelf.

5. Field conditions

These are the REAL WORLD CONDITIONS under which your project has to work and has to be demonstrated.

	Real world field condition 1	Real world field condition 2	Real world field condition 3
Field condition requirement. In which field conditions does the system have to operate? Describe the one, two or three most important field conditions.	The system must be able to perform multi-object tracking on at least 3 people in a large room with a clear line of sight to each person. The system should act as an autonomous security system.	Outside without direct sunlight facing the camera lens the system should be able to detect people and the bachelor.	In a paintball arena for the game, the Bachelor Gauntlet, the turret will be positioned to face where the bachelor enters the course and has a line of sight of the bachelor from start to finish of the course.

6. Student tasks

6.1 Design and implementation tasks

List your primary design and implementation tasks in bullet list format (5-10 bullets). These are *not* product requirements, but *your* tasks.

1) A suitable 2-axis gimbal chassis needs to be constructed to hold the weight of the paintball gun.
2) The suitable motors and servo needs to be selected to move the gun and pull the trigger respectively.
3) The 3D printed gun mount, flexible finger and motor mounts need to be designed and printed.
4) A suitable camera needs to be found and mounted on the gun to provide a point of view (POV) site for capturing images to send to the AI.
5) A processing platform equipped with a GPU powerful enough to do all the graphics processing needs to be acquired, and integration with the system must be implemented.
6) An effective AI for object recognition, and object tracking system needs to be designed and implemented.
7) The PID with suitable overshoot for rudimentary predictive aim needs to be designed and implemented.
8) Image preprocessing before being sent to an object detection system algorithm needs to be implemented.
9) Motor drivers need to be designed, tested and placed on PCBs for the gimbal motors and servo.

6.2 New knowledge to be acquired

Describe what the theoretical foundation to the project is, and which new knowledge you will acquire (*beyond* that covered in any other undergraduate modules).

The foundation of the project is integration of AI computer vision with control systems on an embedded platform. The new knowledge to be learnt through the course of this project includes but not limited to:

1) The student will learn how to perform image sampling and preprocessing, e.g. scaling, edge detection, grey style conversion.
2) The student will learn how to make an image recognition AI with motion prediction.
3) The student will learn how to design and implement a 2-axis PID gimbal controller.
4) The student will learn how to perform accurate motor control using PWM and H-bridge design and implementation.
5) The student will learn how to use the processing platform and GPU chosen.
6) The student will acquire knowledge on how to tune the PID and train the AI model.

Part 3. Main Report

1. Literature study

To design the proposed system effectively, it is essential to understand its specific requirements. The system can be split into two subsystems, the computer vision subsystem, and the control subsystem with hardware. The computer vision subsystems must be designed for the live video stream input and therefore must be fast. The computer vision subsystem must also be as accurate as possible since the output determines if an object is targeted, but the complexity that is introduced by increasing the accuracy is limited by the first requirement for speed.

1.1 Computer vision solution

Many suitable algorithms exist for the computer vision subsystem, but the following are of interest due to their real-time video application capabilities.

The Fast R-CNN (Region-based Convolutional Neural Network) and Faster R-CNN as discussed in [5], which indicates that the Faster R-CNN framework was an improvement upon the fast R-CNN and is a competitive option for real-time multi-object tracking. However, the YOLO (You Only Look Once) algorithm, as shown in [1], balances speed and accuracy making it suitable for live video streaming and is therefore a better solution than Faster RCNN.

The Single Shot MultiBox Detector (SSD) algorithm which is based on a deep convolutional neural network (CNN), produces a set of fixed boundary boxes and associated scores indicating the presence of instances of object classes in the boxes. The SSD algorithm eliminates proposal generation and subsequent pixel or feature resampling stages, which significantly improves the speed as shown in [6]. Similar to the YOLO algorithm the SSD only requires one pass through the neural network (NN) in order to classify objects, making it well-suited for real-time computer vision applications.

The YOLO algorithm is one of the newest approaches and is the preferred choice for live video stream applications since only one pass through the network is required for each frame, reducing the time complexity. The YOLO algorithm can be used in conjunction with a Kalman filter to detect objects and then track them as shown in [2].

The YOLO algorithm operates by fetching an image from the video stream, resizing the image to a fixed size (608x608), and normalising all the pixels to a range [0,1]. The image is passed through a convolutional neural network (CNN) for feature extraction. YOLO uses a series of convolutional and pooling layers. The image is divided into a grid of size QxQ (YOLOv3 is 13x13), with each grid cell predicting a fixed number of bounding boxes, (YOLOv3 is 3). For each bounding box, the following metrics are acquired: the centre coordinates (relative to the grid cell); width and height of the bounding box; confidence score of an object inside the bounding box; and class probabilities for the object. Non-Maximum Suppression (NMS) removes duplicate bounding boxes of the same object by removing the bounding boxes with the lowest confidence scores. A Kalman filter can then predict the position of objects in the next frame.

The choice of the framework is guided by the results in [5] and [7], which demonstrate YOLO's superior speed. The YOLO framework relies on a CNN and the design is explored in [5] where the authors designed two models, namely, the YOLO model, and the fast YOLO model. The fast YOLO model is the same as the YOLO model except it uses nine convolutional layers instead of 24, and fewer filters in those layers. The smaller CNN is better suited for this project since the

processing platform's resources are limited and speed is a primary requirement.

1.1.1 CNN

The main component of the YOLO algorithms is the CNN. To design a custom version of the YOLO algorithm a deep understanding of how CNNs work is needed and the contribution of each component. In [8] the basic structure and inner workings were clarified. The article explains the benefits of using a CNN compared to traditional ANNs, which have been implemented in prior modules (basic implementation). The article emphasises that convolutions are effective for problems where features are not spatially dependent. This means that the position of the object within the passed-in frame does not have an effect on the detection.

The input to the CNN can be multi-dimensional which is needed for images since one needs to pass in an image of 416x416 pixels and each pixel has Red, Green, and Blue (RGB) values. Therefore, the input is 416x416x3 shape as shown below in Figure 1.

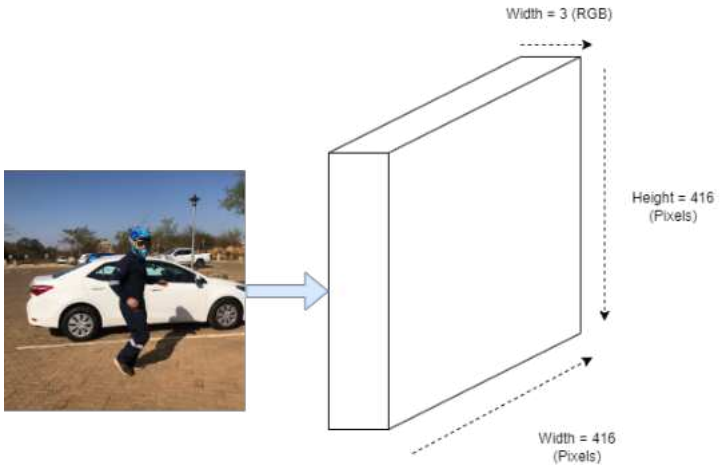


Figure 1.
Input layer dimensions

The biggest difference in the CNN structure is that instead of using fully connected layers, where every input is connected to every neuron, convolutions are used which look at local regions in the input using a sliding window. This is much more efficient and better mimics how visual processing works biologically.

The other benefit of using convolutions is the hierarchical feature extraction. This means each convolutional layer can be associated with different filters. Therefore, we can extract different features from the given image/input. Figure 2 shows how the different filters extract different features when convolved with the passed-in image/input. This creates automatic feature extraction. Other approaches require manual feature extraction, an example is edge detection, normally the input image is converted to greyscale and the highest values bounding pixels are detected using gradient calculations and an outline 'edge' of an object can be extracted.

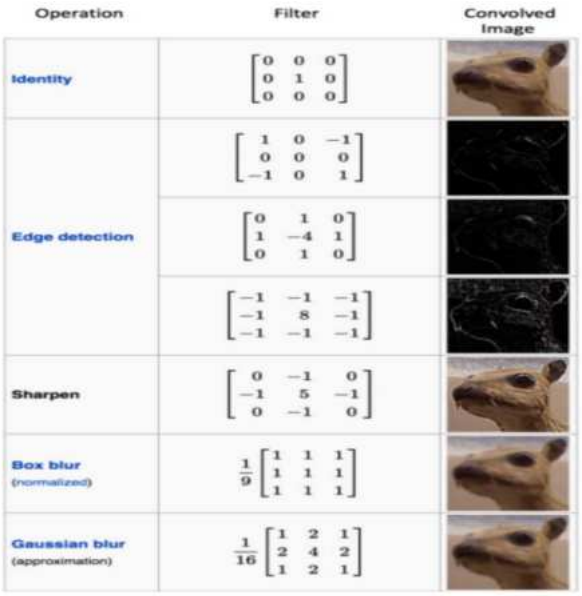


Figure 2.
Different filter convolutions from [8]

when considering how Yolo predicts bounding boxes, one can see a problem before even trying to implement it. The problem is that for each cell in the image, three bounding boxes are being generated for objects and the cells do not communicate what each of them has predicted before the output stage, so you each up with multitudes of bounding boxes being displayed for each of the objects in the image. The way to handle this is to use Non-Maximum Suppression (NMS) as explained in [3]. This uses the Intersection Over Union (IoU) metric to detect which bounding boxes are referring to the same object. It then only displays a single box for each object. The box that is displayed is the box with the highest confidence.

Figure 3.
Non-maximum suppression image taken from [3]

The IoU algorithm which NMS, loss and other measurement metrics use is a mathematical expression for accurately representing the alignment of two bounding boxes. In [4], IoU is

explained for both logical and mathematical processes, and the use of IoU is explained for the loss function in a neural network which is also examined in [9].

1.2 PID and hardware

The gun is used to shoot the Bachelor object moving through a designated area. The control sub-system needs to be designed to ensure the gun is positioned correctly and fast enough to achieve a successful shot. The control sub-system incorporates an overshoot with a fast settling time to compensate for the time of flight of the paintball. The smooth movement of the turret will come from the control sub-system which uses a Proportional–Integral–Derivative (PID) algorithm that takes in the current position of the turret, the next set point required, and time, as inputs and then calculates the movement required to match the designed reaction. MATLAB is used to complete the PID design along with the various tuning methods shown in [10].

The intent is to design and implement a fast YOLO algorithm to work with the live video stream to identify and differentiate between ‘friendly’ people and ‘unfriendly’/ ’The Bachelor’. The research performed in [7] will be used to achieve the required speed, by altering characteristics such as the size of the CNN input layer (pixels), the number of convolutional layers, and filters. The computer vision sub-system must be fast enough to attain the required five frames per second. The centre position of ’the Bachelor’ is used as the input set point for the PID controller which will be tuned using the methods discussed in [10].

1.3 Contribution

The CNN is designed to incorporate as many of the tools and techniques that are used in the YOLOv3 model as shown in [1], and [7] so comparisons can be made. This includes using similar convolutions, filters, NMS, and IoU implementations, but the structure has been reinvented since none of the heavily optimised mathematical and logical libraries were used. This means the speed requirement forces major downsizing designs for the model to meet the requirement on the Jetson Nano processing platform.

The mathematical operations and logic from [4], [9], and [3] are used in the design of the custom implementations.

Although YOLOv3 is a special kind of CNN, a lot of the implementation principles came from [10] that remain unchanged from the theory and most of the implementation theory from [1], and [7] had to be changed to suit the new YOLOv3-tiny version design.

A very small version of the YOLOv3 model is optimised for speed and designed to be run on the Jetson Nano processing platform. This system can identify the bachelor, person (’friendly’/not targets), car, and dog classes. The only classes of interest are the person and bachelor, but the other two classes are added to show the system can identify other objects. The dog class specifically because the dog and person class can be predicted as one another in most computer vision projects when the person is close to the camera because the hair on the head can look similar to a dog. The computer vision subsystem communicates with the control subsystem via serial communications to control the gun. The PID for the control sub-system is designed using techniques from [10].

2. Approach

2.1 Hardware

The hardware design may not be the most technical challenge in the system, but if not done correctly, it could lead to the whole system failing.

The design for the turret required that the turret be able to manoeuvre a gun system 180° in both the x and y rotational axis. The first step was acquiring the gun and taking measurements. Since the system is intended to be used at paintball arenas, the gun selected had to be a generic shape compatible with what most paintball game institutions would have available. Once the gun was acquired and the measurements were taken, the design became simple. A high torque servo was placed as close to the CG as possible on both planes (x and y) of the gun and moved outwards to allow for tolerance of the size of the guns. The 'U' shape of the turret support was then designed around the gun and the three servos. After a basic design was developed, the dynamic movement of the structure had to be considered. Bearings, couplers, and supports were designed to reduce friction and remove the load from the servos. A large ball bearing had to be designed for the base to supply the required support and move-ability.

The turret had to manoeuvre an approximately 7 kg gun system 180° in both the x and y rotational axis. This means the turret had to be designed to hold the weight of the system but without being too heavy and exceeding the torque limitations of the servo at the base. The choice of material was between 304 stainless steel sheet metal that could be bent and drilled, or a 3D-printed plastic structure which could be printed in parts allowing the structure to be modular. The 3D printing was chosen since a home-built 3D printer was available. The choice of plastic meant the structure would be light enough to use relatively small servos and the production cost was a lot less than the alternative.

The PCB was later added to reduce clutter and provide surge protection. This made the system more modular and easier to assemble/disassemble for transport. The LCD display was chosen to be used as a tool for debugging the PID system on the microcontroller.

2.2 Software

The computer vision system had to be designed with three main characteristics in mind. The system had to be fast. One of the most important specs was the speed requirement of at least 5 FPS. This meant the first step would be to research all fast computer vision approaches, keeping in mind that the results shown in research papers would be far above the achievable results on the Jetson Nano processing platform. From the research, the most promising seemed to be the template-matching approach. This is an extremely simple process where an image template is used to slide over the entire image from the video stream and calculate the similarity between the template and the window section on the image.

The second characteristic to design for was generalisation. This is where the template matching option became unsuitable. Template matching was unsuitable since it cannot generalise, and is very unreliable under different lighting conditions and angles. The solution was to research CNN-based solutions. However, the problem with this solution was that it introduced the third

design characteristic, accuracy.

Template matching was very accurate, but with the CNN-based solutions, the accuracy would be lower. After some research, the YOLO algorithm proved to be the best solution since it was incredibly fast and accurate. The YOLOv3 model seemed like a good solution since it was no longer simply a CNN architecture and the concepts seemed logical. YOLOv3 proved to be capable of real-time detection in all research papers, achieving speeds of up to 30 FPS. YOLOv3 predicts bounding boxes coordinates and class probabilities in a single pass reducing the load on resources. YOLOv3 also uses multi-scale detection. This means passed-in image arrays are split into cells and the object detection is then performed in each cell. The image is split into a different number of cells at different scales. The multi-scale prediction allows YOLOv3 to predict objects that are far from and close to the camera. YOLOV3 also uses predefined anchor boxes to better detect objects of different sizes. The benefit of using YOLOv3, which can perform at high frame rates, is the ability to perform tracking-by-detection, which is perfect for aiming a gun.

The only problem with the YOLOv3 model was discovered after implementing it on the Jetson Nano processing platform. The YOLOv3 model is super fast (15-25 FPS) on a high-performance GPU system, like the one in the development laptop being used (Nvidia RTX 2060). However, on the Jetson Nano's limited resource GPU (128-core Nvidia Maxwell) the model struggled to get close to 1 FPS. This was expected, so the solution was to trade off the accuracy for speed, and reduce the size of the model significantly. Following the design of the YOLOv3-Tiny architecture the 5 FPS spec was achieved.

3. Design and implementation

3.1 Design summary

This section summarises the project design tasks and how they were implemented (see Table 1).

Deliverable or task	**Implementation**	**Completion of deliverable or task, and section in the report**
Model Architecture	The model architecture was designed to be faster than YOLOv3 and YOLOv3-tiny to be able to meet the speed requirements on the processing platform with limited resources.	Completed Section 3.4.1
Configuration	The system was optimised in training and running by selecting predefined training sets, image size, classes, learning rate, and anchor boxes.	Completed Section 3.6
Image Processing.	Resize images, convert to RGB, normalize pixel values, colour mask, and apply data augmentations.	Completed Section 3.4.2
Forward Pass.	Input goes through convolutional layers, two scale predictions are made, and each scale predicts 3 bounding boxes per cell.	Completed Section 3.4.3
Loss Function.	During training the loss is calculated for object loss, box coordinates, class, and no object.	Completed Section 3.4.4
Filtering results.	NMS with threshold is implemented which uses IOU which is also implemented.	Completed Section 3.4.5
Yolo executor with real-time detection and visual analyzer.	A script designed to call on different functions in order to perform real-time object tracking by detection and visually display it.	Completed Section 3.4.6

Utilities functions.	Loading/saving checkpoints, data loading and reprocessing, and bounding box conversions.	Completed Section 3.4.7
PCB.	Printed circuit board design and fabrication process from scratch.	Completed Section 3.5.1
PID.	PID simulation on MATLAB and implementation on ESP32.	Completed Section 3.4.8
Turret.	Gun turret and components designed and simulated in Solid Edge. Designs 3D printed, assembled and tested.	Completed Section 3.5.2

Table 1.
Design summary

3.2 System block diagram

From the system block diagram in the project proposal, the updated high-level functional block diagram is shown below in Figure 4.

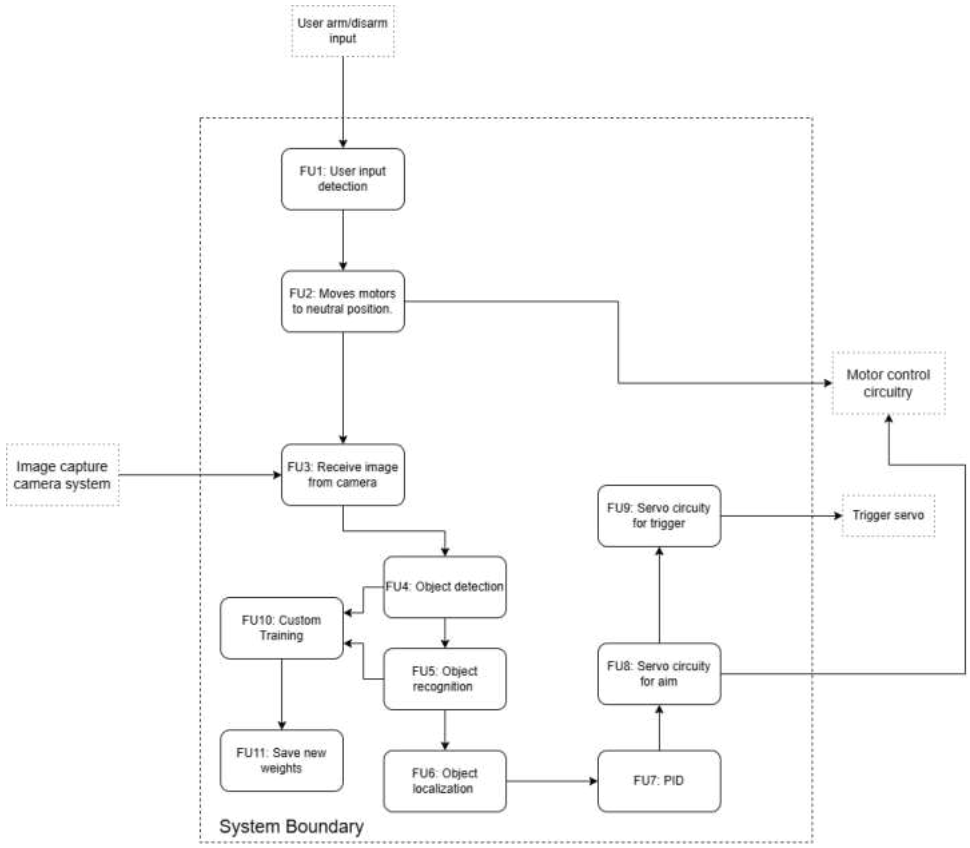


Figure 4.
High-level project functional block diagram

The systems functional block diagram has not changed much (on a high-level) from the project proposal. The system can detect the user's input using FU1 (User input detection). FU2 (Move motors to neutral position) and FU3 (Move motors to search) both move the motors without any detection of the Bachelor. FU2 is used to set the original position of the gun at 90° on the x and y axis. FU3 is used to slowly rotate the gun when searching for the Bachelor. The images are received from the USB web camera using FU4 (Receive image from camera). FU5 (Object detection), FU6(Object recognition), and FU7(Object localisation) are used to identify the Bachelor and track the object. The coordinates of the Bachelor are sent to the PID controller (FU8), which then controls the turret to aim the gun at the Bachelor. FU9 (Servo circuitry for aim), and FU10 (Servo circuitry for trigger) are used for controlling the servos to position the gun and pulling the trigger. In the event that more training is needed because the suit being used for the Bachelor is slightly different from the one that was used for training the network, FU11 (Custom training) and FU12 (Save new weights) can be used to continue training the network on new images for a few epochs.

The functional block diagram shown below in Figure 5 shows more detail than 4. This is still a high-level view of all the main functions in the system but it explains the system in more detail.

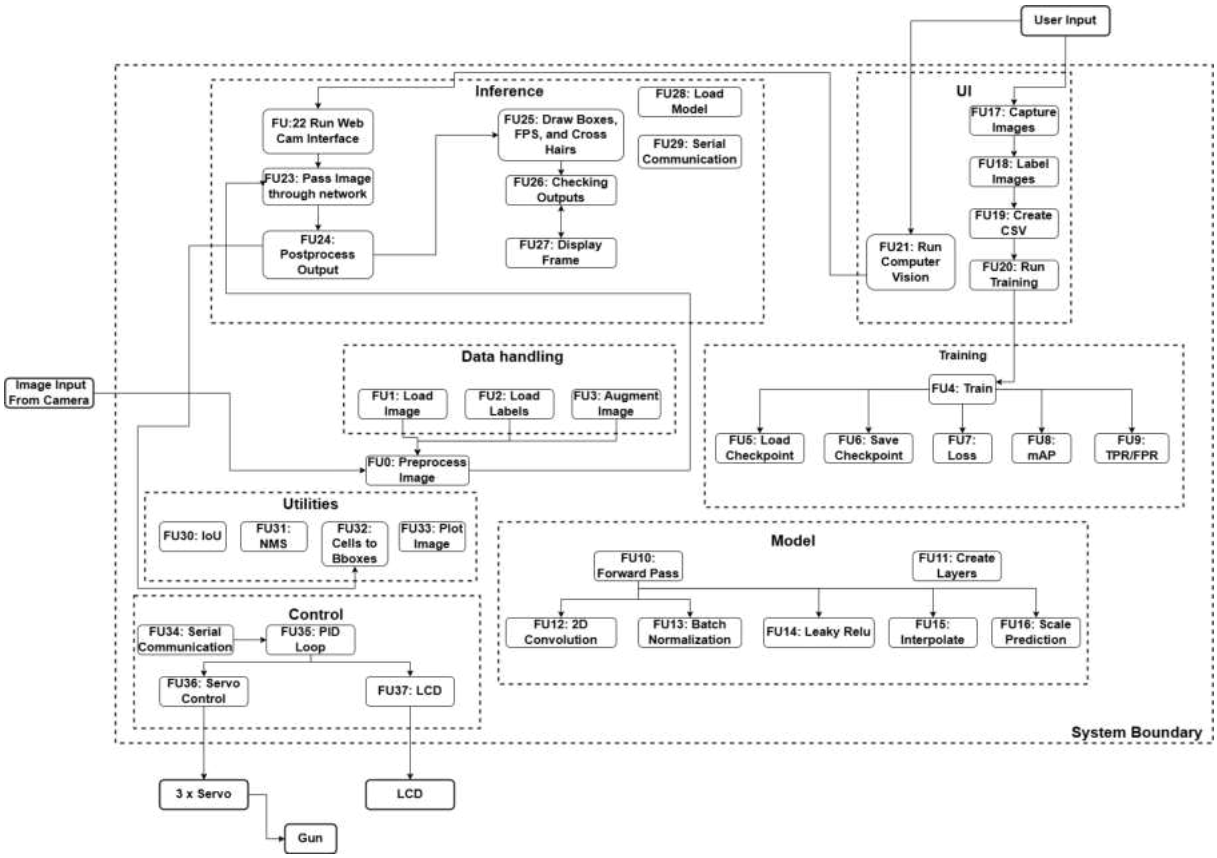


Figure 5.
Project functional block diagram

FU0 (Prepossess Image) ensures the correct format of the JPG which should be RGB, adds padding and normalises all values from 0 to 1. The images either come from the web camera interface or after going through FU1 (Load Image) in the dataset which just fetches an image from a training or testing dataset. FU2 (Load Labels) executes at the same time as FU1 when fetching items from a database for training. FU3 (Augment Image) is used when training on a dataset, the image and label are shifted and colour jitter is applied to the image to ensure the model is not being trained on exactly the same images every epoch to avoid overfitting. FU4 (Train) is used for adjusting the weights within the filters, FU5 (Load Checkpoint) loads the saved weights into the model filters, and FU6 (Save Checkpoint) saves or updates the weights file once training has been completed. During training FU7 (Loss) is used to calculate the inaccuracy in the class and bounding box predictions, FU8 (mean average precision (mAP)) is used to calculate the mean average precision using the testing dataset, and FU9(TPR/FPR) is used to show how often the class prediction is correct for certain classes. For running the computer vision application and displaying the output, FU10 (Forward Pass) is used to pass the prepossessed image through the CNN. The CNN was created using FU11 (Create Layers). FU12 (2D Convolution) adds padding and uses a sliding window approach to slide a kernel (filter) over the 2D input data, performing element-wise multiplication and summing the results which produces a feature map to highlight certain features in the input. During training, FU13 (Batch Normalisation) normalises the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation which allows for higher learning rates. FU14 (Leaky Relu) is the activation function which is used after FU12 and FU13 in order to introduce non-linearity to the model. This allows the

model to learn complex patterns. During upsampling, the FU15 (Interpolate) function is used to combine feature maps of different scales. This increases the spatial dimensions of feature maps, which produces up-sampled feature maps with increased spatial dimensions. The Scale Prediction (FU16), is used to generate the final output tensors for the different scales. It applies a series of convolution layers to the feature map in order to predict bounding boxes, objectness scores, and class probabilities. The system is run on the Jetson Nano and the programs are run by using the UI over a remote desktop setup. The user selects to do custom training or to run the turret (FU21 Run Computer Vision). When the user selects to train the system, the system captures images (FU17), presents the images to the user to label (FU18), and then creates training and testing CSV files of the images and labels (FU19). The system then trains the baseline network with the CSV files just created for limited epochs and only certain layers (FU20). The inference script is used to pass images through the YOLO CNN. It first sets up and runs the webcam interface (FU22), and whenever possible the image from the webcam is passed through the network (FU23). The outputs from the CNN require post-processing (FU24) before being used to draw the bonding boxes (FU25). Since the system operates a gun, there are checks and safety nets in place to control the final sections (FU26: Checking Outputs). Finally, the frame is displayed using FU27: Display Frame. The inference script is also responsible for loading the model's weights FU28, and communicating with the control sub-system using serial communications FU29. The IoU function (FU30) is used to generate a metric to reflect how boxes intersect. NMS (FU31) uses the IoU function to determine which bounding boxes to keep from the predictions generated from the CNN output. During the post-process function already mentioned, the cells to bounding boxes function (FU32) is used to convert the predictions per cell to bounding boxes. A function was created to view individual images with boxes (FU33). On the control sub-system, serial communications (FU34) are used to get set points from the Jetson. These set points are fed into the PID controller loop (FU35). Within the loop, the servos are controlled (FU37) and diagnostic information is displayed on the LCD (FU38).

One main change was made from the initial system block diagram. Previously the idea was to learn to recognise certain people, for example, 'Mark' would walk in front of the system and the system would take images of 'Mark' and add him to a database of 'friendly' people. The obvious problem with this approach is it could be used for a system that uses template matching or similar approaches, but the CNN's goal is to generalise objects. Therefore, if one trains a CNN to identify what a person is on a dataset of 10000 images of people, and then train it on 100 images of 'mark', the system will identify 'mark' as a person object instead of a mark object since it has learnt to generalise the features of a person which 'mark' fits. The solution to this is to only train for 'Mark' and not train on the 10000 images of people. However, then anyone who enters the camera's field of view would most likely be identified as 'Mark' since the system has never been trained on anyone else's images. The improved system design classifies the object as a bachelor or a person class based on a uniform chosen for the Bachelor.

3.3 Theoretical Background

The main active component of the project is the computer vision system which has been designed and implemented. To be able to understand what has been done, we need to first briefly explain what computer vision is. Computer vision is a field of artificial intelligence that aims to enable computers to interpret visual data from a camera in a similar way that humans do, and extract useful information from the images to then act upon.

The main aspects of any computer vision program involve image acquisition, image processing, feature extraction, object recognition, and for this project object localisation. Image acquisition and processing are handled in a later section but feature extraction, object recognition and object localisation are all handled by the Convolutional Neural Network (CNN). CNNs are an artificial network with nodes which represent neurons. The structure allows them to be very good at processing grid-like information which makes them perfect for handling images. The basic building block of the CNN is the convolution operation which is performed between weights and inputs, the standard convolution operation is defined by equation (1) below.

$$(f * g)(p) = \sum_{s+t=p} f(s)g(t) = \sum_{s} f(s)g(p-s) \tag{1}$$

For a given convolution layer the output is a feature map, which is automatic feature extraction through the neural network. The feature map can be computed as shown in equation (2) below where $\hat{a}l_{i,j,k}$ is the activation at position (i,j) in feature map k of layer l, $\hat{w}l_{r,s,c,k}$ is the weight at position (r,s) connecting feature map c to k, $\hat{b}l_k$ is the bias for feature map k, and f is the activation function.

$$\hat{a}l_{i,j,k} = f\left(\sum_{r,s,c} \hat{w}l_{r,s,c,k} \cdot \hat{a}l - 1_{i+r,j+s,c} + \hat{b}l_k\right) \tag{2}$$

Typical neural networks can be described by equation (3).

$$\text{Input} \xrightarrow{\text{Conv}} \text{ReLU} \xrightarrow{\text{Pool}} \cdots \xrightarrow{\text{FC}} \text{Output} \tag{3}$$

In this project, down-sampling convolution layers were used, presented by equation (4) where s is the stride value, instead of pooling layers which are represented by equation (5). The benefit of this is that convolution is a dynamic method where optimal weights are learned as opposed to pooling which just uses the max value (max pooling) or average, this means the filters adapt the down-sampling behaviour to the data. The main problem with pooling is that it discards information, but the convolution uses all the data points which means it makes better use of the available information. Using strided convolution layers for down-sampling also means fewer layers in the network which means faster forward pass. The graphical comparison between these two methods is shown in Figure 6.

$$y_{ij} = \sum_{m} \sum_{n} w_{mn} x_{s \cdot i+m, s \cdot j+n} + b \tag{4}$$

$$y_{ij} = \max_{(p,q) \in R_{ij}} x_{pq} \tag{5}$$

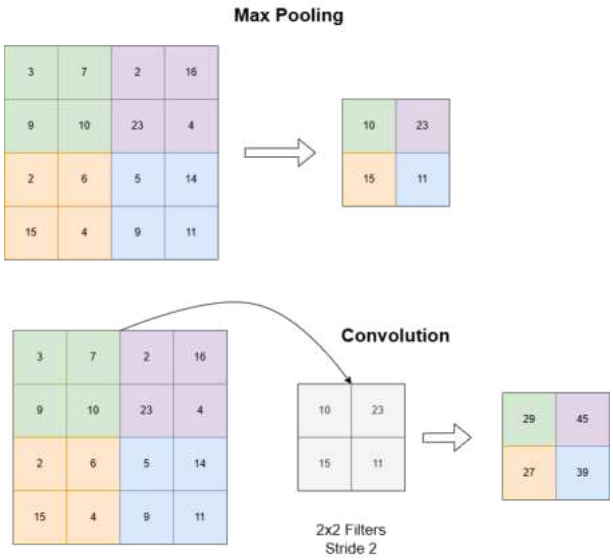


Figure 6.
Max pooling vs convolution

3.4 Software design

3.4.1 Model Architecture

To design the model architecture an examination of the standard YOLOv3 architecture had to be conducted. YOLOv3 was chosen as the benchmark from the literature study, 1. YOLOv3 uses Darknet-53 as a backbone and then uses an additional 53 layers coming to a total of 106 layers. Before testing began it was known that this was far too large to achieve the 5 FPS needed. The architecture of the YOLOv3 model was so large it required calculating hypothetical speeds that could be achieved for a forward pass and then work back to calculate the size of the network that would be required in order to reach the 5 FPS needed.

YOLOv3 tiny can achieve 0.5 FPS on the Jetson Nano and it has a model size of 236 MB and a total FLOPs (Floating Point Operations) count of 65.86 GFLOPs. In comparison the standard YOLOv3-tiny model is 33.7 MB, with 5.56 GFLOPs and can achieve 5 FPS on the Jetson Nano. Both these models are highly optimised and used libraries to improve their efficiency which are not available to me. It was predicted that whatever was implemented would be able to achieve 75% of the theoretical speed.

The model was designed to maintain the same scale (number of cells and spatial dimensions) as were present in the YOLOv3-tiny model, so the following 11 convolutional layer model architecture was designed as shown below in Figure 7.

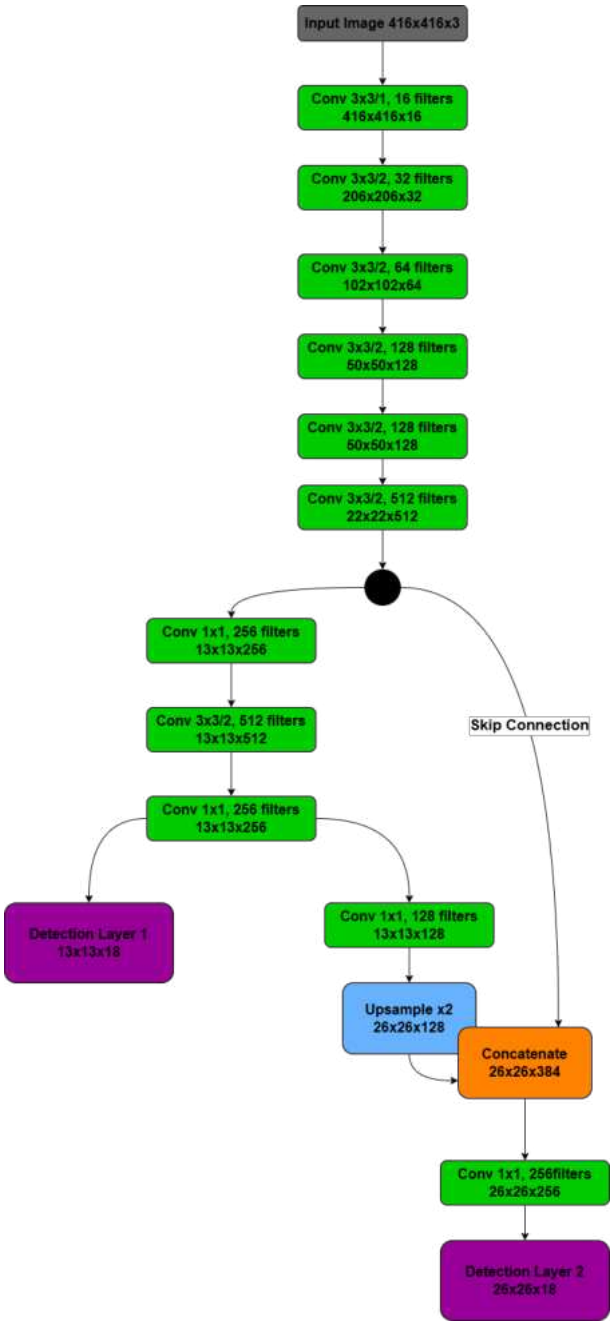


Figure 7.
Reduced YOLOv3-tiny CNN architecture

The model size and FLOPs were calculated. The equations below 6, assume the 32-bit float (4 Bytes) per parameter.

$$Model\,Size(Bytes) = Sum(parameters\,per\,layer) * 4 \tag{6}$$

$$FLOPs = 2 * H * W * Cin * Cout * K^2 \tag{7}$$

For the FLOPs calculation, H is the output height, W is the output width, Cin is the number of inputs, $Cout$ is the number of outputs, and K is the kernel size.

$$FPS_{estimate} = \frac{GPU\,Performance_{adjusted}}{Model\,FLOPs} \tag{8}$$

The $GPU\ Performance_{adjusted}$ is the recorded performance of the 128-core NVIDIA Maxwell GPU using my code which does not make use of highly optimised GPU libraries. This is 4.46 GFLOPs, this is explained below.

$$Memopry\,Usage = Model\,Size + (Largest\,Activation\,Size * 4) \tag{9}$$

After getting the values for both the YOLOv3 and YOLOv3-tiny, the results were compared to the known values of the YOLOv3 model and they were within 10% of the know sizes. The known size of the YOLOv3 model is 246 MB and the calculations produce a size of 236 MB. The smaller size can be accounted for due to parameters outsize of standard layer architecture that were not accounted for. The results are accurate enough though to estimate performance. After performing the calculations for the full 106-layer YOLOv3 and the tiny version (16 layers) I worked backwards and found that 12 convolution layers would meet the 5 FPS requirement. The calculations for the 12 layers can be seen below. Layer 12 is the from scale the final scale prediction layer.

$$\begin{aligned}
Layer\,1 &: (3*3*3*16+16)*4 = 1792\,bytes \\
Layer\,2 &: (3*3*16*32+32)*4 = 18560\,bytes \\
Layer\,3 &: (3*3*32*64+64)*4 = 73984\,bytes \\
Layer\,4 &: (3*3*64*128+128)*4 = 295424\,bytes \\
Layer\,5 &: (3*3*128*256+256)*4 = 1180672\,bytes \\
Layer\,6 &: (3*3*256*512+512)*4 = 4720640\,bytes \\
Layer\,7 &: (1*1*512*256+256)*4 = 525312\,bytes \\
Layer\,8 &: (3*3*256*512+512)*4 = 4720640\,bytes \\
Layer\,9 &: (1*1*512*256+256)*4 = 525312\,bytes \\
Layer\,10 &: (1*1*256*128+128)*4 = 131584\,bytes \\
Layer\,11 &: (3*3*384*256+256)*4 = 393728\,bytes \\
Layer\,12 &: (1*1*256*18)*4 = 18432\,bytes \\
TOTAL &: 12606080\,bytes \approx 12.6MB
\end{aligned} \tag{10}$$

The FLOPs are calculated using Equation (7). The calculation for each layer is shown below in (11).

$$
\begin{aligned}
Layer\,1 &: 2*416*416*3*16*3^2 = 149520384 \\
Layer\,2 &: 2*208*208*16*32*3^2 = 398721024 \\
Layer\,3 &: 2*104*104*32*64*3^2 = 398721024 \\
Layer\,4 &: 2*52*52*64*128*3^2 = 398721024 \\
Layer\,5 &: 2*26*26*128*256*3^2 = 398721024 \\
Layer\,6 &: 2*13*13*256*512*3^2 = 398721024 \\
Layer\,7 &: 2*13*13*512*256*1^2 = 44302336 \\
Layer\,8 &: 2*13*13*256*512*3^2 = 398721024 \\
Layer\,9 &: 2*13*13*512*256*1^2 = 44302336 \\
Layer\,10 &: 2*26*26*256*128*1^2 = 44302336 \\
Layer\,11 &: 2*26*26*384*256*3^2 = 1196163072 \\
Layer\,12 &: 2*26*26*256*27*1^2 = 9345024 \\
TOTAL &: 3880261632 \approx 3.88 GFLOPs
\end{aligned}
\tag{11}
$$

Since the Jetson Nano can run the YOLOv3-tiny model at 5 FPS which has 5.56 GFLOPs and this new model has 3.089 FLOPs, if one assumes a direct linear relationship (it is not linear, but close enough to estimate nearby extrapolation), then the theoretical FPS for the new model would be $5 * \frac{5.56}{3.88} = 7.16$ FPS. But remembering that this is just the theoretical calculation using only the theoretical number of calculations and assuming perfectly optimised code, so using the previously defined realistic assumption of 75%, the model should achieve 5.37 FPS.

3.4.2 Image processing

Before using the images captured by the web camera it is important to do some prepossessing. This is absolutely critical as the CNN is designed for a specific-sized image to be passed through and the raw images captured by the webcam could be far too large and require excessive memory. The pseudo-code below (Algorithm 1) describes the image preprocessing before being passed through the CNN.

Algorithm 1 Image Preprocessing Function

tensor_batch ← PreprocessImage(image, target_size)

1: *resized* ← resize(image, (target_size, target_size)) {Resize image to square 416x416}
2: **if** channels(resized) = 3 **then**
3: *rgb* ← BGR2RGB(resized) {Convert from BGR to RGB color space}
4: **end if**
5: *normalized* ← *rgb*/255.0 {Normalize pixel values to [0,1] range}
6: *transposed* ← transpose(normalized, (2,0,1)) {Reorder from HWC to CHW}
7: *tensor* ← toTensor(transposed) {Convert to tensor}
8: *tensor_batch* ← unsqueeze(tensor, 0) {Add batch dimension}
9: **return** *tensor_batch*

Resizing the image can make enormous performance differences in forward pass speed especially

if high-resolution cameras are being used. The camera in the system is a Logitec C310. This camera has a 15 MP (1920 x 1080 pixels) resolution. If one calculates the memory usage knowing that for each pixel there are 3 channels (RGB) which are each represented by a float32 object (4 bytes).

$$\begin{aligned} \text{Original Image: } & 1920 \times 1080 \\ \text{Each Pixel } & = 3 \times 4 \\ \text{Memory Required } & = \text{width} \times \text{height} \times 12 \\ \text{Memory Required } & = 1920 \times 1080 \times 12 = 24.88 \text{ MB} \\ \text{Memory Required After Resizing } & = 416 \times 416 \times 12 = 2.08 \text{ MB} \end{aligned} \tag{12}$$

This is around a 12 times reduction in memory usage per frame, freeing up resources for other processes. The smaller images mean much faster convolutions since the output size is smaller. The Resizing is mainly because the CNN structure is fixed and designed around the image size. Each layer has a fixed input size and the feature map calculations depend on the input size. The equation below (13) shows the output dimension formula used for each convolution layer to produce the output size. W is the input size, K is the kernel size, P is the padding, and S is the stride.

$$\text{Output Dimension} = \left(\frac{W - K + 2P}{S}\right) + 1 \tag{13}$$

NOTE: Pay attention to how stride affects the output size, Section 3.3 explains how this is used for down-sampling and used in place of pooling.

The second part of the image processing occurs at the end of passing the frame through the CNN and after getting the predicted bounding boxes which have been scaled to the image that is about to be displayed and just before any instructions are sent to the control sub_system. A final check needs to be done on the predictions as it is crucial that no unintended targets are mislabelled as the Bachelor. Since the CNN is very small (only 11 layers) the amount of feature extraction that has been completed is limited, therefore, a final check was added. If any bonding box that is about to be displayed is the bachelor class, a blue colour mask is created, the image is converted to HSV and the bonding box is divided into 3 vertical ROIs. After applying the blue colour masks to the three ROIs, the colour composition of the object being detected can be ensured that it matches the blue colour composition of what the Bachelor should be. The pseudo-code below (Algorithm 2) describes this check before sending the coordinates of the Bachelor to the control sub-system.

Algorithm 2 Colour Verification Analysis Function

$is\_blue \leftarrow$ ColorThreshold(image, bbox, blue_threshold)

$height, width \leftarrow$ image.shape {Get image dimensions}
$x1, y1, x2, y2 \leftarrow$ bbox {Extract bbox coordinates}
$x1 \leftarrow \max(0, \min(x1, width - 1))$ {Clip coordinates to image bounds}
$y1 \leftarrow \max(0, \min(y1, height - 1))$
$x2 \leftarrow \max(0, \min(x2, width - 1))$
$y2 \leftarrow \max(0, \min(y2, height - 1))$
if $x1 \geq x2$ OR $y1 \geq y2$ **then**
    return False {Invalid bounding box}
end if
$total\_height \leftarrow y2 - y1$ {Calculate section heights}
$section\_height \leftarrow total\_height/3$
$top\_thresh \leftarrow blue\_threshold - 0.02$ {Define section thresholds}
$mid\_thresh \leftarrow blue\_threshold$
$bot\_thresh \leftarrow blue\_threshold - 0.1$
$sections \leftarrow [(y1, y1 + section\_height), (y1 + section\_height, y1 + 2 * section\_height), (y1 + 2 * section\_height, y2)]$
$thresholds \leftarrow [top\_thresh, mid\_thresh, bot\_thresh]$
for all $(i, (y\_start, y\_end))$ in $sections$ **do**
    $roi \leftarrow image[y\_start : y\_end, x1 : x2]$ {Extract region of interest}
    $hsv \leftarrow$ BGR2HSV(roi) {Convert to HSV color space}
    $mask \leftarrow$ inRange(hsv, blue_low, blue_high) {Create blue mask}
    $percentage \leftarrow$ sum(mask)$/(roi.width * roi.height)$
    if $percentage < thresholds[i]$ **then**
        return False {Section fails threshold check}
    end if
end for
return True

3.4.3 Forward pass

From Figure 7, this architecture also shows how the input image is passed through the CNN. Each convolution layer serves a specific purpose and extracts different features. The first layers extract basic features like edges, the second to the sixth layer down-sample to get finer and finer features, and then the flow splits into a large object detection scale prediction and a small object detection scale prediction. The large scale-prediction uses large cells breaking the input into big cells (13x13), and the small object detection uses a finer grid (26x26), the smaller cells allow it to detect smaller objects. If one inspects the dimensions of each layer one will notice that during the down-sampling layers (where stride is 2) the spatial dimensions are reduced. This is to account for the increase in feature channels. The same computational load is required for each of these layers because the spatial dimensions are reduced, if they were not and the number of channels was increased, the computational load for each consecutive layer would grow.

The main part of the forward pass is the 2D convolution that is performed between the input layer and the filter which is a matrix of weights. This procedure is crucial for having an effect on

CNN. The pseudo-code below in Algorithm 3 explains how the convolutions work.

Algorithm 3 Conv2D Implementation

$output \leftarrow$ Conv2D(input, weight, bias, stride, padding, dilation)

$batch\_size, channels, in\_height, in_width \leftarrow$ input.shape
$out\_channels, in\_channels, kernel\_h, kernel\_w \leftarrow$ weight.shape
if $padding > 0$ **then**
    $padded \leftarrow \text{zeros}(batch\_size, channels, in\_height + 2padding, in_width + 2padding)$
    $padded[:, :, padding : in\_height + padding, padding : in\_width + padding] \leftarrow input$
    $input \leftarrow padded$
end if
$out\_height \leftarrow (in\_height + 2padding - dilation(kernel\_h - 1) - 1)/stride + 1$
$out\_width \leftarrow (in\_width + 2padding - dilation(kernel\_w - 1) - 1)/stride + 1$
$kernel\_indices \leftarrow$ create kernel indices$(kernel_h, kernel_w, dilation)$
$output\_indices \leftarrow$ create output indices$(out_height, out_width, stride)$
$unfolded \leftarrow$ extract patches$(input, kernel_indices, output_indices)$
$weight\_matrix \leftarrow \text{reshape}(weight, (out\_channels, -1))$
$output \leftarrow$ batch matmul$(weight_matrix, unfolded)$
if *bias* is not None **then**
    $output \leftarrow output + bias.view(1, -1, 1, 1)$
end if
$output \leftarrow \text{reshape}(output, (batch\_size, out\_channels, out\_height, out\_width))$
return output

The pseudo-code above is visualised in the diagram below in Figure 8.

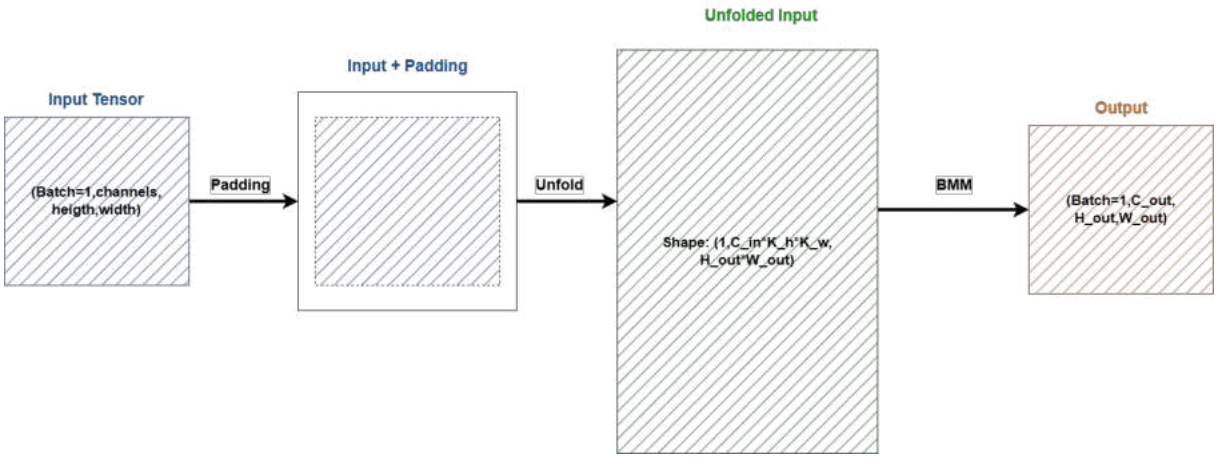


Figure 8.
High-level diagram of convolution in forward pass

Working through the diagram in Figure 8 from left to right the following section will explain each operation, starting with the image padding.

The simplest way to explain how padding is implemented is described by the equation (14) below, where the original pixels of the image are relocated to the correct position within the new image

and all the pixels outside the region are given the value 0.

$$I_{padded}(i,j) = \begin{cases} I(i-P, j-P) & \text{if } P \le i < H+P \text{ and } P \le j < W+P \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

Below is a simplified visual representation in equation (15).

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix} \xrightarrow{\text{pad}(P=1)} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 0 \\ 0 & 5 & 6 & 7 & 8 & 0 \\ 0 & 9 & 10 & 11 & 12 & 0 \\ 0 & 13 & 14 & 15 & 16 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{15}$$

Note that the coordinates indexes referred to in the following explanation start from 0. For coordinates (2,2) in the padded image:

- i=2, j=2 satisfies ≤i<H+P and P≤j<W+P (where P=1)
- Therefore I_padded(2,2) = I(2-1,2-1) = I(1,1) = 6

For coordinate (0,0) in padded image:

- i=0, j=0 does not satisfy the condition
- Therefore I_padded(0,0) = 0

The first logical question to ask is why should the images be padded? This is done for three main reasons. The first reason is because the output after every convolution is smaller than the input, meaning the feature maps will shrink since the output of the first layer is the input to the second, so depending on the padding used, the padding can preserve the spatial dimensions. This is shown in Figure 9. The reason for padding in this project is not to preserve the spacial dimensions but rather for the benefit of the next two reasons. The second reason for downsampling is somewhat intuitive since it's easy to visualize. Picture the input layer as a grid and the filter as a smaller grid which is placed at the top left of the big input grid and multiplied with the section of the big grid that overlaps with it, and then it is shifted two positions to the right and the process is repeated again and again. One can now understand how all the pixels in the middle area of the input are used multiple times while the border pixels are only used in the multiplication once. This is why one adds padding, so the edge pixels are no longer actually the edge and the edge features are not missed. The third reason is for better feature extraction since the important edge features are lost, this also provides better context.

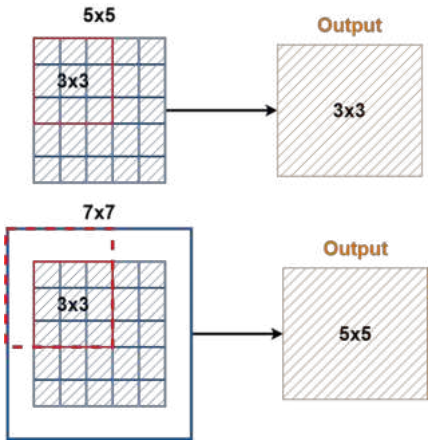


Figure 9.
Padding effect

The next step is to unfold the padded image. By unfolding the padded image the padded image is broken up into multiple patches. The thing to remember is the stride. If the kernel/filter has dimensions 3x3 with a stride of 2, then the first patch's third column of pixels will be the same as the second patch's first column. To get these patches of the input a sliding window was used, a visual representation of this unfolding is shown below in Figure 10.

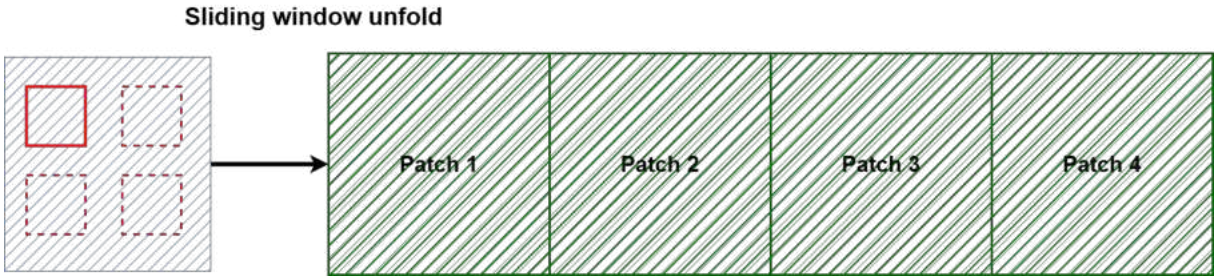


Figure 10.
Diagram showing how the sliding window is used to unfold

An example of the unfolding is shown below in Figure 11

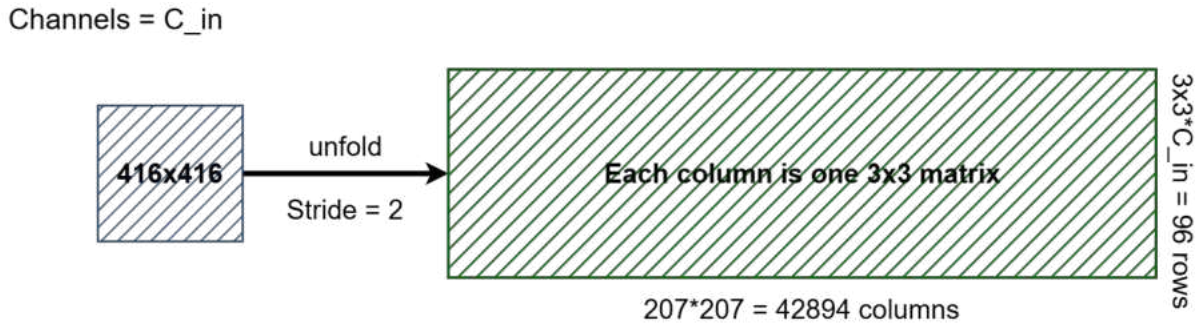


Figure 11.
Example of an unfold operation

$$
\begin{aligned}
\text{H\_out: } & \frac{\text{H\_in-K+2P}}{S}+1 \\
\text{W\_out: } & \frac{\text{W\_in-K+2P}}{S}+1 \\
\text{H\_in=W\_in} &= 416 \\
K &= 3 \\
S &= 2 \\
P &= 0 \\
\text{H\_out=W\_out=} & \frac{416-3+2(0)}{2}+1=207
\end{aligned}
\tag{16}
$$

Once the input has been padded and unfolded to form patches, one needs to perform the convolution which involves the matrix multiplication of the filter across all the patches and summing the results to form an output, Figure 12 is a graphical illustration of this.

Figure 12.
Diagram explaining BMM

Algorithm 4 Batch Matrix Multiplication in Conv2D

1: $W_{reshape} \leftarrow \text{reshape}(W, (C_{out}, C_{in}K_hK_w))$ {Reshape weights}
2: $W_{batch} \leftarrow \text{expand}(W_{reshape}, (B, C_{out}, C_{in}K_hK_w))$ {Replicate for batch}
3: $X_{unfold} \leftarrow \text{unfold}(X, (B, C_{in}K_hK_w, H_{out}W_{out}))$ {Unfolded input}
4: $Output \leftarrow \text{BatchMatMul}(W_{batch}, X_{unfold})$ {BMM operation}
5: $Output \leftarrow \text{reshape}(Output, (B, C_{out}, H_{out}, W_{out}))$ {Final reshape}

This can be explained by explaining the convolution process. Below in equation (17) is the mathematical expression for the input dimensions which have already been covered but this is the mathematical definition, W is the weights and X_{unfold} is the unfolded input. Remember, this model was designed for both training and inference and that is why the batch parameter is included in all these calculations. During training different batch sizes can be used to move many images from the training dataset through the network at once but in the inference script, the batch is always one since just one frame is fetched from the web camera at a time.

$$\begin{aligned} W_{batch} &\in \mathbb{R}^{B \times C_{out} \times (C_{in}K_hK_w)} \\ X_{unfold} &\in \mathbb{R}^{B \times (C_{in}K_hK_w) \times (H_{out}W_{out})} \end{aligned} \tag{17}$$

The simple equation 18 below shows that the weights are simply matrix multiplied with the input, during training, this happens for each batch but for inference just once.

$$Output_b = W_b \times X_b \tag{18}$$

Below in equation (19) I show the matrix multiplication for a single 3x3 filter and a 3x3 patch.

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \odot \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} = \begin{bmatrix} x_{11}w_{11} & x_{12}w_{12} & x_{13}w_{13} \\ x_{21}w_{21} & x_{22}w_{22} & x_{23}w_{23} \\ x_{31}w_{31} & x_{32}w_{32} & x_{33}w_{33} \end{bmatrix} \tag{19}$$

The output is then the element-wise multiplication and summation plus a bias as shown in (20) below.

$$y = \sum_{i=1}^{3}\sum_{j=1}^{3} x_{ij}w_{ij} + b = (x_{11}w_{11} + x_{12}w_{12} + x_{13}w_{13} + x_{21}w_{21} + x_{22}w_{22} + x_{23}w_{23} + x_{31}w_{31} + x_{32}w_{32} + x_{33}w_{33}) + b \tag{20}$$

The bias is added to shift the output away from zero. This helps handle inputs that are not centred around zero increasing the flexibility of the feature detection. The bias also acts as a threshold for the activation (when neurons "fire"). Without the bias, the output would always be zero for an input of zero. The whole process is repeated for each input channel. The results are summed

across channels. This is the key for feature extraction since each output channel has its own filters and bias, all of them together create a feature map. This is the same logic but repeated for the batches (21) when training.

$$\begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{ik} \end{bmatrix} b \times \begin{bmatrix} b11 & \cdots & b_{1j} \\ \vdots & \ddots & \vdots \\ b_{k1} & \cdots & b_{kj} \end{bmatrix}_b = \begin{bmatrix} c_{11} & \cdots & c_{1j} \\ \vdots & \ddots & \vdots \\ c_{i1} & \cdots & c_{ij} \end{bmatrix}_b \tag{21}$$

The next step is batch normalisation, the purpose is to normalise the layer input during training and inference. During training the input is normalised to the batch mean and variance (described in section 3.3), during inference the input is normalised using the running statistics. The equation below (22, 22) shows how the input is normalised for the inference operation.

$$y = \gamma \frac{x - \mu_{running}}{\sqrt{\sigma^2_{running} + \varepsilon}} + \beta \tag{22}$$

$$\begin{aligned} \mu_{running} &= (1 - \text{momentum}) \cdot \mu_{running} + \text{momentum} \cdot \mu_B \\ \sigma^2_{running} &= (1 - \text{momentum}) \cdot \sigma^2_{running} + \text{momentum} \cdot \sigma^2_B \end{aligned} \tag{23}$$

The value of of γ is originally set to 1 and β is set to 0. The momentum (α) is set to 0.1, ε is set to 1×10^{-5} and then β and γ are updated as shown below in equation (24), where L is the loss, η is the learning rate.

$$\begin{aligned} \gamma &\leftarrow \gamma - \eta \frac{\partial L}{\partial \gamma} \\ \beta &\leftarrow \beta - \eta \frac{\partial L}{\partial \beta} \end{aligned} \tag{24}$$

The final step is the activation function. The activation function is a very simple and powerful equation which introduces non-linearity into the system. This basically means taking an input and mapping it to an activated output. Without this step one could have layers that just perform matrix multiplication. The most common function is the ReLU. ReLU is just 0 for any negative inputs and then the input for anything not negative is simply the value of the input with a minimum value of 0. Most examples found of computer vision CNN models use leaky ReLU since it prevents 'dying ReLU', which is when inputs are consistently negative and the output is always zero so no gradient is produced which is needed for training. Leaky ReLU is basically the same as ReLU but instead of being zero for negative input, it has a small gradient of 0.01. The equation for ReLU is shown below in equation (25), and leaky ReLU in equation (26) where α is 0.01.

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{25}$$

$$\text{Leaky ReLU}(xw + b) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases} \tag{26}$$

The output is then sent to the post-process function to extract all the information and eventually display it. Figure 16 below summarises what the forward pass does in the context of the system.

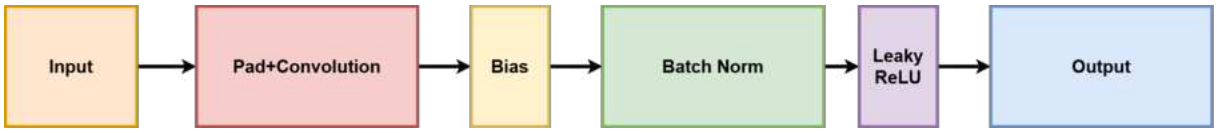


Figure 13.
Forward pass summary

3.4.4 Loss function

The loss function acts as a feedback mechanism during training. The model is trained using supervised learning which means the input image is forward passed through the network to generate predictions, the loss function is then passed the predictions and the known correct labels and boxes, and the loss function generates a value to reflect the error in the prediction. This shows how inaccurate the model is, and is used to calculate gradients that update the weights which are then used in the next iteration and the process repeats. The diagram below in Figure 14 shows this process.

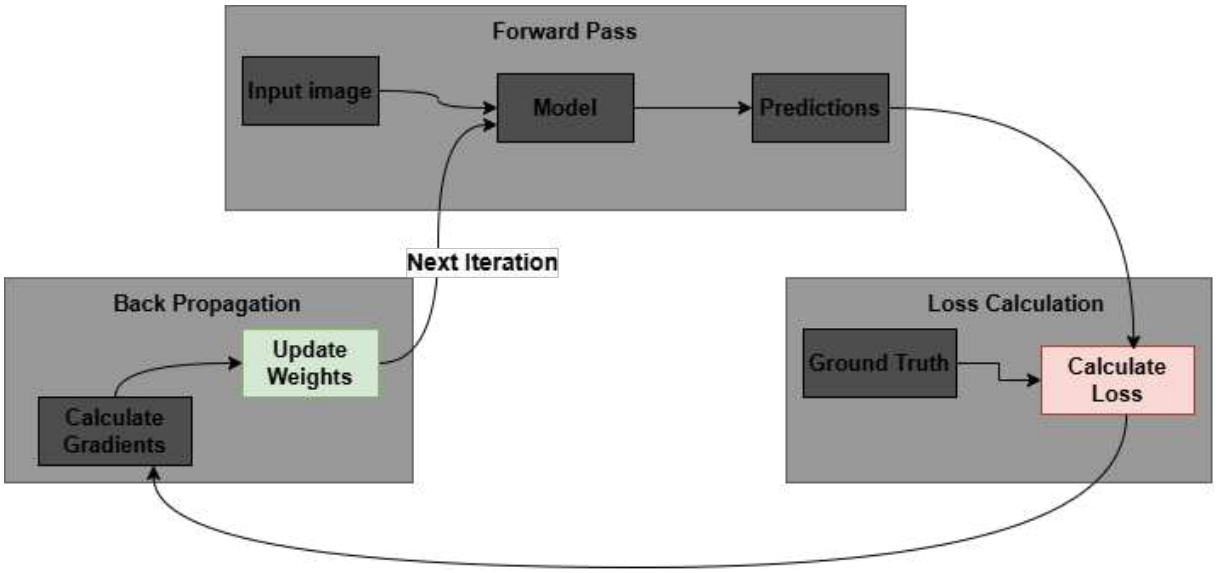


Figure 14.
Diagram to show loss function in the training loop

The loss function breaks down the prediction into three main components, object confidence, box predictions, and class predictions. The object confidence is used to calculate the no-object loss and the object loss. The box coordinates are used to calculate the box loss, and the class predictions are used to calculate the class loss.

The object confidence loss is calculated using Binary Cross Entropy (BCE) loss which is defined by equation (27) below where $\sigma(C)$ is sigmoid of predicted confidence, IoU is the intersection over union with the target box. The standard equation for BCE is shown in equation (28). The output is then weighted by $\lambda_{obj} = 10$.

$$L_{obj} = BCE(\sigma(C), IoU) \tag{27}$$

$$BCE = -(y\log(p) + (1-y)\log(1-p)) \tag{28}$$

The BCE formula is also used for the no-object loss calculation but applied where no object exists. The formula for the no object loss is shown in equation (29) below. This is then weighted by $\lambda_{noobj} = 5$. This is how false positives are penalised.

$$L_{noobj} = BCE(\sigma(C), 0) \tag{29}$$

The box loss has to use a different equation obviously and for this, the Mean Squared Error (MSE) was used. For the x and y coordinates the values are passed through the sigmoid activation function, $\sigma(t_x)$, $\sigma(t_y)$. The formula for the x and y loss is shown below in equation (30), where $\hat{x}$, $\hat{y}$ are the target coordinates. The MSE equation for the x and y coordinates is shown in equation (31).

$$L_{xy} = MSE(\sigma(t_x), \hat{x}) + MSE(\sigma(t_y), \hat{y}) \tag{30}$$

$$L_{box_{xy}} = \frac{1}{n}\sum_{i=1}^{n}[(\sigma(t_x)_i - \hat{x}_i)^2 + (\sigma(t_y)_i - \hat{y}_i)^2] \tag{31}$$

The box loss for height and width is similar but the activation is the exponential function ($\exp(t_w)$, $\exp(t_h)$) instead of the sigmoid. The equation for the box loss of width and height is shown below in equation (32) where $\hat{w}$, $\hat{h}$ are the target dimensions. The two box losses are then combined and weighted by $\lambda_{box} = 8$. The MSE equation for the width and height dimensions is shown in equation (33).

$$L_{wh} = MSE(\exp(t_w), \hat{w}) + MSE(\exp(t_h), \hat{h}) \tag{32}$$

$$L_{box_{wh}} = \frac{1}{n}\sum_{i=1}^{n}[(\exp(t_w)_i - \hat{w}_i)^2 + (\exp(t_h)_i - \hat{h}_i)^2] \tag{33}$$

The class loss uses cross-entropy loss with class weights. The equation used to calculate the class loss is shown below in equation (34) where w_c is the class weight (5.0 for bachelor and person, 1.0 for car and dog), y_c is the true class label (one-hot), and p_c is the predicted class probability. The final output is then weighted by $\lambda_{class} = 10$.

$$L_{class} = \sum_{c}(w_c \cdot y_c \cdot \log(p_c)) \tag{34}$$

Please note that the equation used for intersection over union (IoU) is given below in equations (35, 42, 37, 43).

$$x_1 = x - \frac{w}{2}, \quad y_1 = y - \frac{h}{2}, \quad x_2 = x + \frac{w}{2}, \quad y_2 = y + \frac{h}{2} \tag{35}$$

$$I = \max(0, \min(box1_{x2}, box2_{x2}) - \max(box1_{x1}, box2_{x1})) \cdot \max(0, \min(box1_{y2}, box2_{y2}) - \max(box1_{y1}, box2_{y1})) \tag{36}$$

$$U = box1_{area} + box2_{area} - I \tag{37}$$

$$IoU = \frac{I}{U + \varepsilon}, \quad \text{where } \varepsilon = 10^{-6} \tag{38}$$

$$L_{total} = \lambda_{box} \cdot L_{box} + \lambda_{obj} \cdot L_{obj} + \lambda_{noobj} \cdot L_{noobj} + \lambda_{class} \cdot L_{class} \tag{39}$$

The diagram below in Figure 15 shows the structure of the loss function.

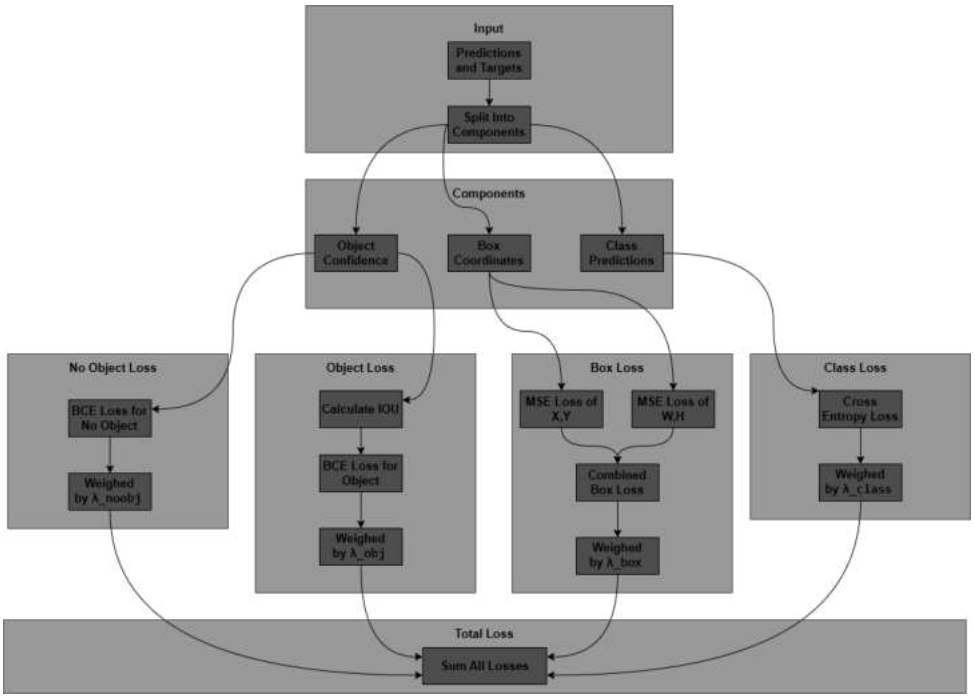


Figure 15.
Diagram to show loss function structure

Table 2 below summarises all the loss parameters used during training. These weights force the model to value certain characteristics and classes more than others.

Parameter	Value	Location	Description
λ_{class}	10	loss.py	Classification loss weight
λ_{noobj}	5	loss.py	No-object loss weight
λ_{obj}	10	loss.py	Object loss weight
λ_{box}	8	loss.py	Box coordinate loss weight
class_weights	[5.0, 5.0, 1.0, 1.0]	loss.py	Per-class weights

Table 2.
Loss Function Parameters

3.4.5 Filtering results

The problem with using YOLO architecture and splitting the input into grid cells where each cell makes two predictions, is dealing with all the predictions at the output. For each object, there may be a multitude of predictions. The solution was to create a Non-Maximum Suppression (NMS) algorithm. The NMS algorithm first applies a confidence threshold across all the predictions, getting rid of any predictions that are not confident enough to proceed to the output. The predictions are then sorted in descending order of confidence. The algorithm then loops through all the predictions starting with the first (highest confidence) and checks for any other predictions that have the same class type, if the Intersection Over Union (IoU) is greater than a certain

threshold (see section 3.6) then the other (lower confidence) prediction is removed. Essentially this starts with the highest confidence prediction for each class (if it exists after the blanket confidence threshold) and then removes all other predictions of the same class that have an IoU that suggests the predictions are referring to the same object. The flow diagram below in Figure 16 is a graphical representation of the NMS process. The boxes list slowly shrinks as boxes are evaluated and either accepted and kept, or removed and the boxes_after_nms list grows as boxes are added until the boxes list is empty and the the boxes_after_nms is returned.

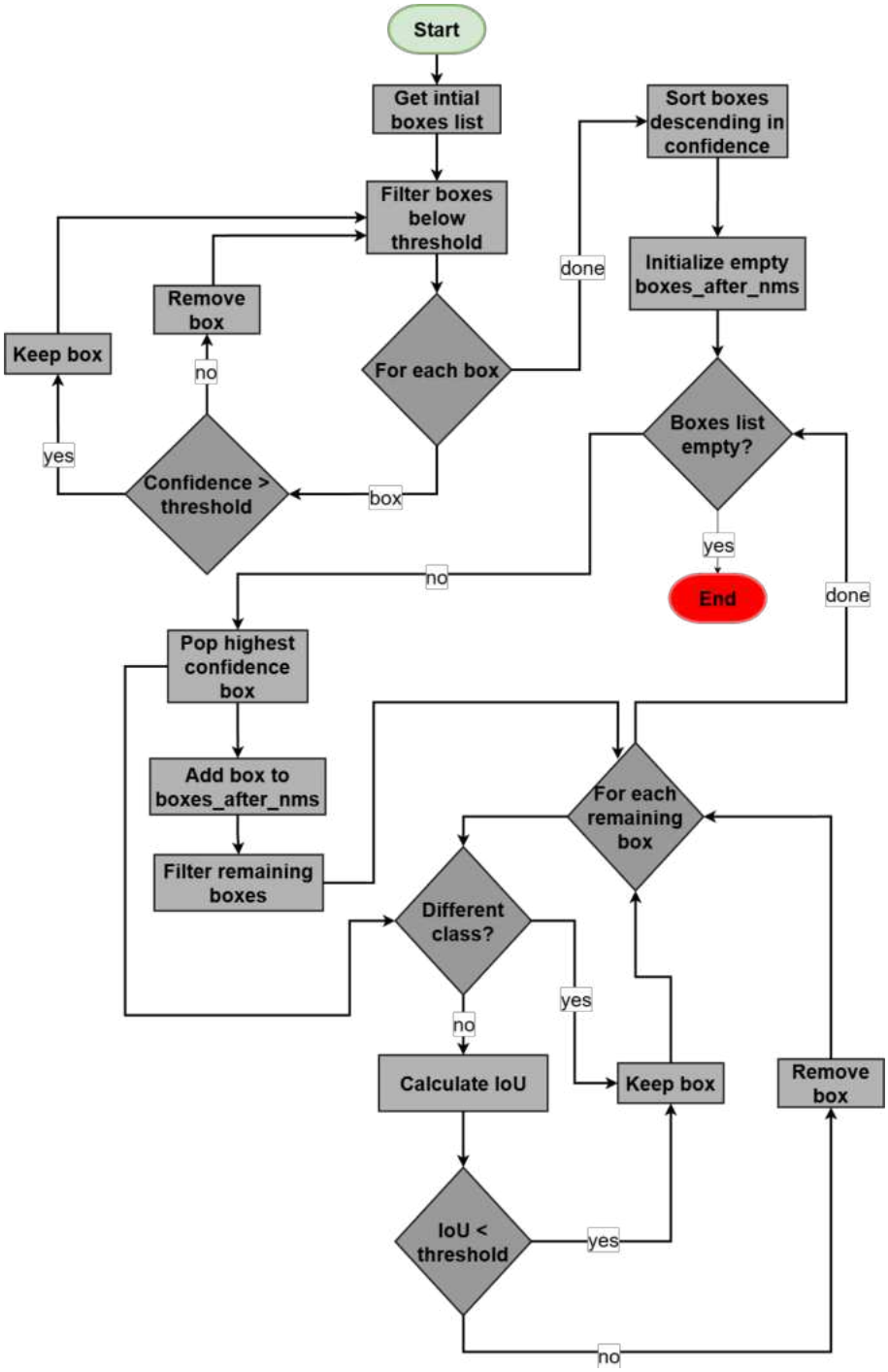
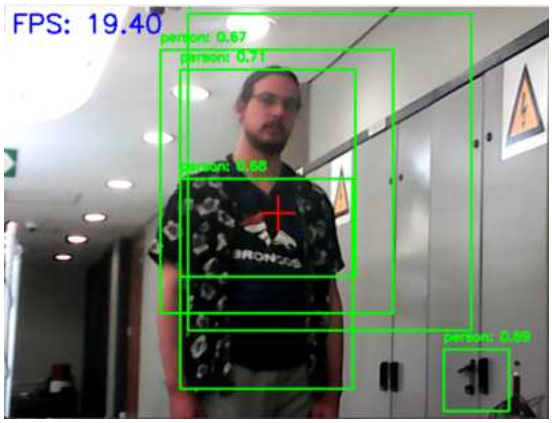
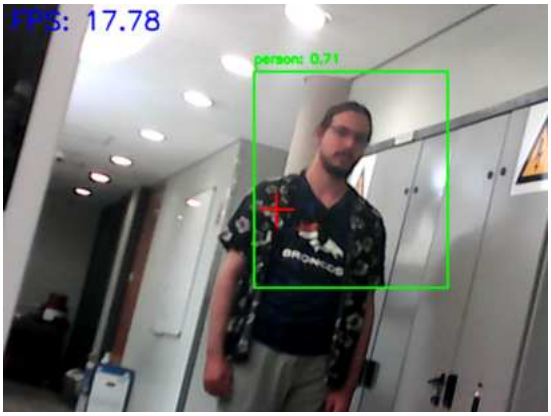


Figure 16.
Flow diagram showing the NMS process

The effect of NMS can clearly be seen in the Figures (17a, 17b) below. A fellow student offered to be part of this demonstration. In Figure 17a the NMS IoU threshold is 0.6 and this means all the bonding boxes that predicted the person and that had an intersection over union value with the box with the highest confidence of less than 0.6 is kept. When the IoU threshold is 0.12, as shown in Figure 17b only a single bounding box is displayed which is what the NMS implementation is designed to do.



(a)
Bounding boxes of a person using NMS IoU threshold of 0.6



(b)
Bounding boxes of a person using NMS IoU threshold of 0.12

Figure 17.
Example of NMS effect

3.4.6 Real-time detection and visualisation

Running the real-time detection and visualisation inference script is a process that contains two main parts. The first part is the initialisation and detection section shown in Algorithm 5. Part 1 is where the model is initialised and the pre-trained weights are loaded and also sets up the counter for frames without detections, as well as the last know direction of the Bachelor. There is also a parameter for changing the last known direction. The next step in Part 1 is to set up the serial communications and enter the main loop. In the main loop, images are captured from the web camera, then pre-processed and sent through the model. The output is then post-processed and sent through NMS. The NMS boxes are then sent through the object analysis section where class, confidence and box are extracted. The offset of the centre of the object from the centre of the frame is also calculated.

Algorithm 5 RunYOLO Real-time Detection - Part 1: Initialization and Detection

$detection\_output \leftarrow$ RunYOLODetection(use_serial)

$cap \leftarrow$ VideoCapture(0)
$model \leftarrow$ LoadModel(checkpoint_file)
$scaled\_anchors \leftarrow$ ScaleAnchors(ANCHORS, S)
$frames\_without\_detection \leftarrow 0$
$last\_direction \leftarrow$ "right"
$direction\_change\_counter \leftarrow 0$
if use_serial **then**
    $ser \leftarrow$ SerialPort(/dev/ttyUSB0, 19200, PARITY_NONE, STOPBITS_ONE)
end if
while $cap.isOpen()$ **do**
    $frame \leftarrow cap.read()$
    $image\_tensor \leftarrow$ PreprocessImage(frame, 416)
    $output \leftarrow model(image\_tensor)$
    $bboxes \leftarrow$ ProcessOutput(output, scaled_anchors)
    $nms\_boxes \leftarrow$ NonMaxSuppression(bboxes, IOU_THRESH=0.12, CONF_THRESH=0.66)
    $frame\_center\_x \leftarrow frame.width/2$
    $frame\_center\_y \leftarrow frame.height/2$
    $objects\_detected \leftarrow$ False
    for all box in nms_boxes **do**
        $objects\_detected \leftarrow$ True
        $class\_pred, confidence, x\_center, y\_center, width, height \leftarrow box$
        if $class\_pred = 0$ **and** $confidence > BACHELOR\_THRESHOLD$ **then**
            $bbox \leftarrow$ CalculateBoundingBox(box, frame)
            $is\_good \leftarrow$ Verification(frame, bbox, BLUE_THRESHOLD)
            if is_good **then**
                $pixel\_x \leftarrow (x\_center * frame.width - frame\_center\_x)/3$
                $pixel\_y \leftarrow (frame\_center\_y - y\_center * frame.height)/3$
            end if
        end if
    end for
end while

Part 2 of the inference script is processing and control which is shown below in Algorithm 6. If the bachelor class is detected as a prediction, the prediction is double-checked to ensure it has a high enough confidence to be acted upon by sending it through a high confidence threshold check. The prediction is then checked to ensure the object that has been detected matches the known colour characteristics of the Bachelor by removing the bonding box area of the image as a region of interest (ROI), dividing it into three sections, and converting the colours to HSV for better colour comparison. If the colour characteristics of the object do not match the known colour percentages of a template Bachelor, then the class is changed to person and the object will not be targeted. If the Bachelor is still detected, the offset of the object from the centre of the frame is calculated, to determine if the crosshairs are within the Bachelor box, and update the last known direction. If the crosshairs are within the box the shoot boolean is set to true. The control section of Part 2 also includes the search algorithm that is used if the system loses the Bachelor. If no objects are detected for 10 frames (2 seconds) the system communicates with the

control system telling it to slowly turn in the last known direction of the Bachelor for 50 frames, and if still no Bachelor is found, then to change direction. The serial communication protocol is x,y,s\n where x is the x offset from Bachelor, y is the y offset and s is a boolean value to shoot or not. The visual feedback for the user displays the image captured from the camera with the post-processed outputs in the form of bounding boxes (green=[person, dog, car], red=[bachelor]), class names and confidence, the frame rate and crosshairs of the gun.

Algorithm 6 RunYOLO Real-time Detection - Part 2: Processing and Control

```
if is_good then
    center_inside ← IsInsideBox(frame_center_x, frame_center_y, bbox)
    last_direction ← "right" if pixel_x > 0 else "left"
    if use_serial then
        data ← FormatData(pixel_x, pixel_y, center_inside)
        ser.write(data.encode())
    end if
else
    class_pred ← 1 {Change to person if not blue}
end if
DrawBoundingBox(frame, bbox, class_pred, confidence)
if NOT objects_detected then
    frames_without_detection ← frames_without_detection + 1
    if frames_without_detection ≥ 10 and use_serial then
        direction_change_counter ← direction_change_counter + 1
        if direction_change_counter ≤ 50 then
            turn_value ← 20 if last_direction = "right" else −20
            ser.write(FormatData(turn_value, 0, 0))
        else
            last_direction ← ReverseDirection(last_direction)
            direction_change_counter ← 0
        end if
    end if
end if
DrawCrosshair(frame)
if KeyPressed("q") then
    cap.release()
    ser.close() if use_serial
    break
end if
```

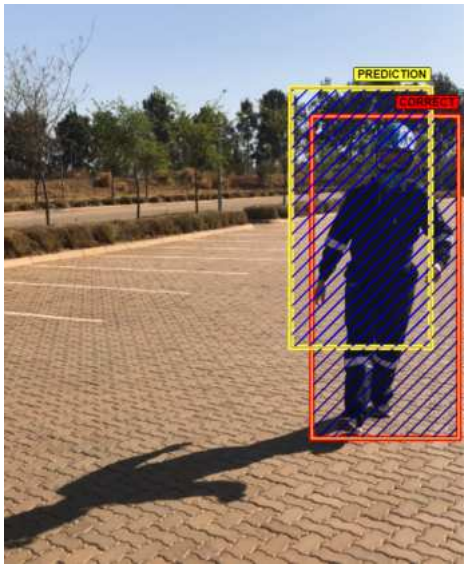
3.4.7 Utilities

The utility functions are functions that are used throughout the project as a tool or measurement metric. These functions have nothing to do with the classes or the flow of operations, so are placed in a separate file to neaten up the other files. The main two utility functions are the IoU function and the cells to bounding boxes (bboxes).

The most important function in the utils.py file is the IoU calculation. This is used to determine

how much two boxes intersect relative to the total area of both boxes. It is used to determine which boxes refer to the same object and is the key to NMS. To better explain this an example is used from a training image to visualize the calculation. If one wants to show how much two boxes overlap one needs to get two areas. Figure 18a shows the intersection, this is the area where both boxes cover. Figure 18b shows the union, this is the combined area of both boxes. We take the intersection area and divide it by the union area to get the overlap relative to the size of the boxes.

(a)
Intersection of two boxes



(b)
Union of two boxes

Figure 18.
Intersection over union

Since width and height are used to represent all box sizes as a standard through the code, a little bit of work is required in order to calculate these areas. For each box, one needs to calculate the top left poison in x and y and the bottom right position in the x and y direction. Equation (40) shows the calculations needed to get these points, this is done for both boxes.

$$\begin{aligned} x_1 &= x - \frac{w}{2} \\ y_1 &= y - \frac{h}{2} \\ x_2 &= x + \frac{w}{2} \\ y_2 &= y + \frac{h}{2} \end{aligned} \tag{40}$$

Once the corner points are calculated for each box the area can be calculated as shown in equation (41) below.

$$\begin{aligned} A_1 &= (box1_{x2} - box1_{x1}) \cdot (box1_{y2} - box1_{y1}) \\ A_2 &= (box2_{x2} - box2_{x1}) \cdot (box2_{y2} - box2_{y1}) \end{aligned} \tag{41}$$

The intersection is then calculated using equation (42) below.

$$I = \max(0, \min(box1_{x2}, box2_{x2}) - \max(box1_{x1}, box2_{x1})) \cdot \max(0, \min(box1_{y2}, box2_{y2}) - \max(box1_{y1}, box2_{y1})) \tag{42}$$

The final calculation is to take the intersection over the union. The union is the combined area of both boxes minus the intersection since it would be added twice otherwise. The equation for IoU is shown in equation (43) below.

$$IoU = \frac{I}{A_1 + A_2 - I} \tag{43}$$

It is mentioned that for scale predictions, each grid cell makes two predictions, so the problem is how are the cell predictions converted to create full bounding boxes? The cells to bounding boxes (bboxes) function converts the grid cell predictions to bounding boxes with coordinates relative to the image. The function is sent the cell format of predictions shape = $(N, A, S, S, C+5)$ where N is the batch size, A is the number of anchors per cell, S is the grid size and C+5 is (objectness, x, y, w, h, class_scores). The initial processing is shown below in equation (44). The function scales prediction by applying anchor dimensions shown in equation (47), ensuring the proper size relationship. The coordinate system uses an activation function, x and y coordinates go through a sigmoid function and the width and height go through the exponential function shown in equation (45). The output is then formatted for the NMS function shown in equation (48).

$$\begin{aligned} \text{BATCH\_SIZE} &= \text{predictions.shape[0]} \\ \text{num\_anchors} &= \text{len(anchors)} \\ \text{box\_predictions} &= \text{predictions[..., 1:5]} \end{aligned} \tag{44}$$

$$\text{If is\_preds:} \begin{cases} b_x, b_y & = \sigma(t_x), \sigma(t_y) \\ b_w, b_h & = \exp(t_w), \exp(t_h) \\ \text{scores} & = \sigma(\text{predictions[..., 0:1]}) \\ \text{best\_class} & = \arg\max(\text{predictions[..., 5:]}) \end{cases} \tag{45}$$

$$\text{cell\_indices} = \begin{bmatrix} 0 & 1 & \cdots & S-1 \\ 0 & 1 & \cdots & S-1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 1 & \cdots & S-1 \end{bmatrix} \tag{46}$$

$$\begin{aligned} x &= \frac{1}{S}(b_x + \text{cell\_indices}_x) \\ y &= \frac{1}{S}(b_y + \text{cell\_indices}_y) \\ w &= \frac{1}{S} b_w \\ h &= \frac{1}{S} b_h \end{aligned} \tag{47}$$

$$\text{output} = [\text{class}, \text{confidence}, x, y, w, h] \tag{48}$$

3.4.8 PID

The PID system is responsible for ensuring the turret moves smoothly, pointing the gun in the correct direction with the correct motion. The system is designed around the response characteristic of the servo motor which is called the plant function. The plant function of a servo motor that moves 60° in 0.2 seconds is known and is shown in equation (49). The coefficients from the transfer function are $a_0 = 1, a_1 = 0.32, b_1 = 0.1$.

$$G(s) = \frac{1}{0.1s^2 + 0.32s + 1} \tag{49}$$

During testing of the servo motors the servos moved too fast and stopped too fast for the hardware. So the PID was designed for a rise time of two seconds and for an overshoot of 5% to account for the moving target. This means the natural frequency (ωn) could be calculated from the rise time as shown in equation (50) below, and the damping ratio (ζ) can be calculated from the percentage overshoot as shown in equation (51) below.

$$\omega_n = \frac{1.8}{t_r} = \frac{1.8}{2} = 0.9 \text{ rad/s} \tag{50}$$

$$\begin{aligned} \zeta &= \frac{-\ln(PO/100)}{\sqrt{\pi^2 + \ln^2(PO/100)}} \\ \zeta &= \frac{-\ln(0.05)}{\sqrt{\pi^2 + \ln^2(0.05)}} \\ \zeta &= \frac{2.996}{\sqrt{18.846}} \\ \zeta &\approx 0.69 \end{aligned} \tag{51}$$

The PID controller parameters can be calculated as shown in equation (52) below. the equation is expanded in equations (53, 54, 55) for $K_p, K_i, and K_d$ respectively.

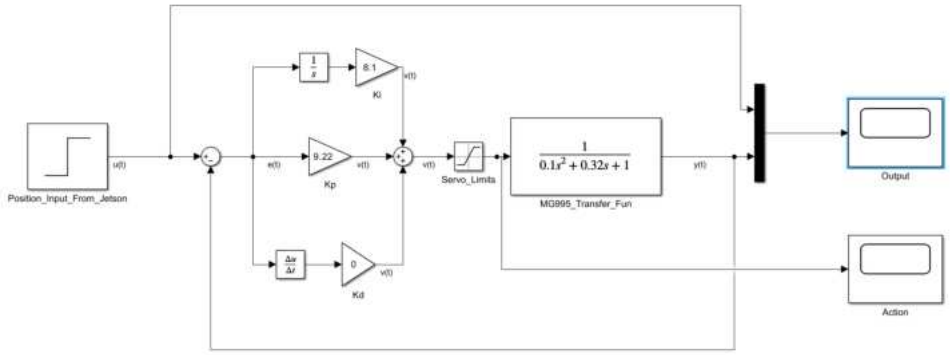
$$\begin{aligned} K_p &= \frac{2\zeta\omega_n - a_1}{b_1} \\ K_i &= \frac{\omega_n^2}{b_1} \\ K_d &= \frac{1 - a_0}{b_1} \end{aligned} \tag{52}$$

$$\begin{aligned} K_p &= \frac{2(0.69)(0.9) - 0.32}{0.1} \\ K_p &\approx 9.22 \end{aligned} \tag{53}$$

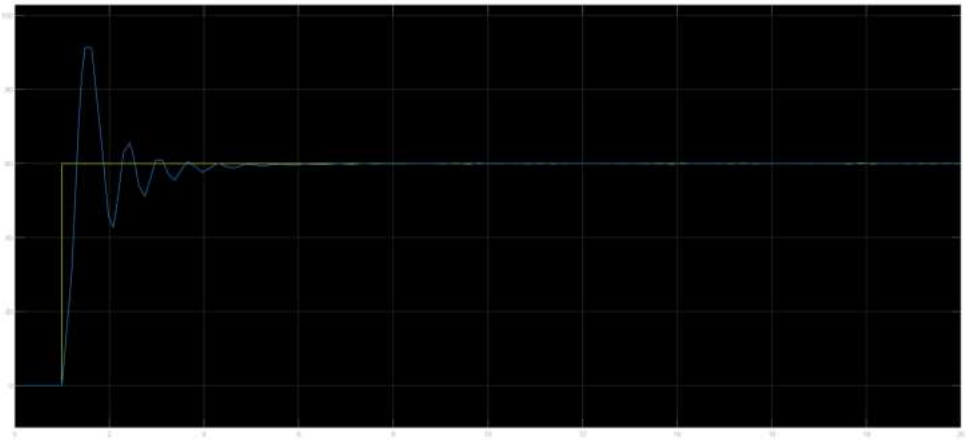
$$\begin{aligned} K_i &= \frac{(0.9)^2}{0.1} \\ K_i &\approx 8.1 \end{aligned} \tag{54}$$

$$\begin{aligned} K_d &= \frac{1 - 1}{0.1} \\ K_d &= 0 \end{aligned} \tag{55}$$

The values were tested using MATLAB software using Simulink as shown in Figures (19a, 19b) below.



(a)
PID Simulink simulation



(b)
PID Simulink simulation output

Figure 19.
PID Simulink simulation

From the output in Figure 19b the response oscillates before settling and therefore, some tuning was required. The oscillating effect during settling was because of the 0 derivative term. If one pictures a car suspension then K_p is the spring providing restoration force, K_i is the car's momentum (used for overshoot) and K_d is the shock absorber which is a dampener. Increasing the K_d term to 1.5 provided a much smoother response as shown in Figure 40 and Figure 41.

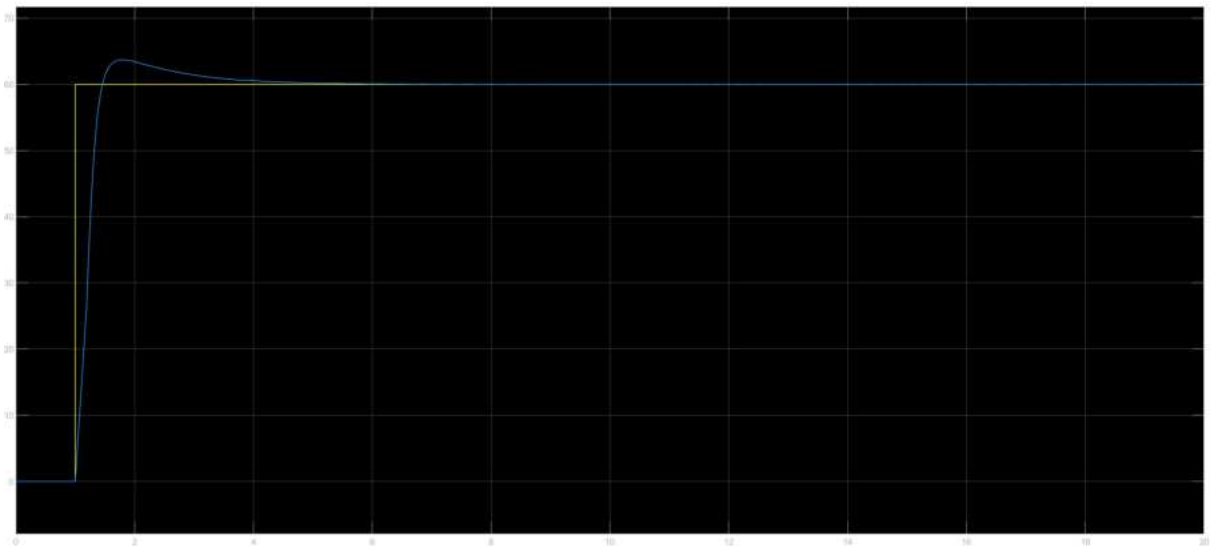


Figure 20.
PID Simulink simulation output after tunning

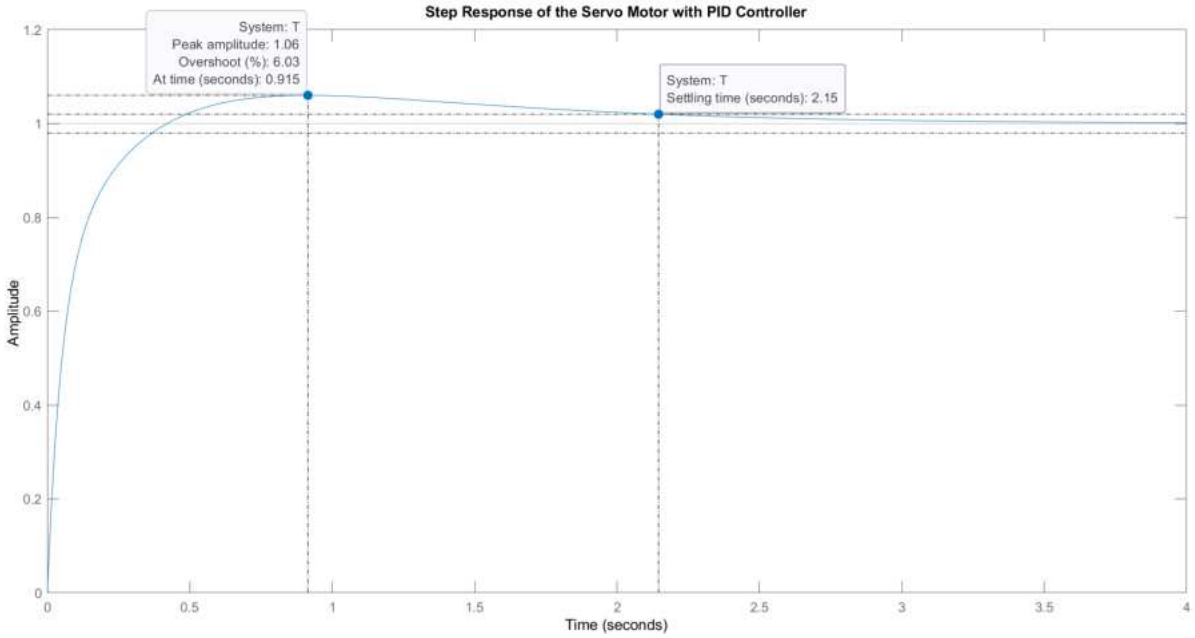


Figure 21.
PID simulation output after tunning

The PID controller is implemented on the ESP32 devkit board as shown in Algorithm 7. The ESP32 with this logic combined with the servos for moving and shooting form the control subsystem of the project.

Algorithm 7 PID Control System for Paintball Gun Targeting

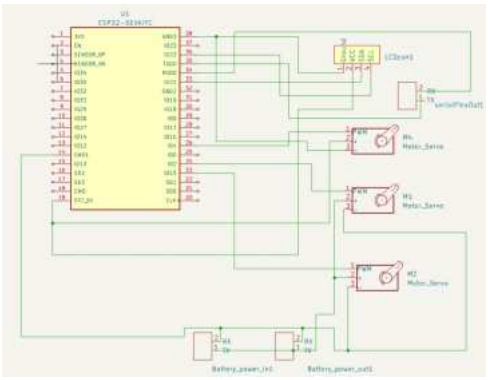
$pid\_control \leftarrow$ InitializePIDSystem()

$Kp \leftarrow 9.22, Ki \leftarrow 8.1, Kd \leftarrow 1.5$ {PID constants}
$max\_integral \leftarrow 50.0, pos\_x \leftarrow 90, pos\_y \leftarrow 90$
$integral\_x \leftarrow 0, integral\_y \leftarrow 0$
$previous\_error\_x \leftarrow 0, previous\_error\_y \leftarrow 0$
InitializeHardware() {Setup servos and LCD}
while true **do**
  if SerialDataAvailable() **then**
    $target\_x, target\_y, shoot \leftarrow$ ParseSerialInput()
    $target\_x \leftarrow target\_x/3, target\_y \leftarrow target\_y/3$ {Scale inputs}
    $error\_x \leftarrow target\_x, error\_y \leftarrow target\_y$
    {Calculate PID terms}
    $elapsed\_time \leftarrow (current\_time - last\_time)/1000.0$
    $P_{out} \leftarrow Kp \times [error\_x, error\_y]$ {Proportional}
    $integral\_x \leftarrow \text{Constrain}(integral\_x + error\_x \times elapsed\_time, -max\_integral, max\_integral)$
    $integral\_y \leftarrow \text{Constrain}(integral\_y + error\_y \times elapsed\_time, -max\_integral, max\_integral)$
    $I_{out} \leftarrow Ki \times [integral\_x, integral\_y]$ {Integral}
    $derivative\_x \leftarrow (error\_x - previous\_error\_x)/elapsed\_time$
    $derivative\_y \leftarrow (error\_y - previous\_error\_y)/elapsed\_time$
    $D_{out} \leftarrow Kd \times [derivative\_x, derivative\_y]$ {Derivative}
    $output \leftarrow P_{out} + I_{out} + D_{out}$ {Combine PID outputs}
    {Apply deadband and constraints}
    if $|error\_x| < 2$ AND $|error\_y| < 2$ **then**
      $output \leftarrow [0, 0]$ {Prevent minor adjustments}
    end if
    $output \leftarrow \text{Constrain}(output, -1, 1)$
    {Update positions and move servos}
    $pos\_x \leftarrow \text{Constrain}(pos\_x + output_x, 0, 180)$
    $pos\_y \leftarrow \text{Constrain}(pos\_y - output_y, 77, 100)$
    ServoX.write(pos_x), ServoY.write(pos_y)
    {Handle shooting mechanism}
    if $shoot = 1$ **then**
      ServoShoot.write(15) {Shooting position}
      Delay(200)
      ServoShoot.write(50) {Rest position}
      $shoot \leftarrow 0$
    end if
    {Update display}
    UpdateLCD($target\_x, target\_y, pos\_x, pos\_y, shoot$)
    {Prepare for next iteration}
    $previous\_error\_x \leftarrow error\_x, previous\_error\_y \leftarrow error\_y$
    $last\_time \leftarrow current\_time$
  end if
  {Gradual error reduction}
  if $|output\_x| > 0$ OR $|output\_y| > 0$ **then**
    $error\_x \leftarrow error\_x - \text{Sign}(error\_x)$
    $error\_y \leftarrow error\_y - \text{Sign}(error\_y)$
  end if
end while

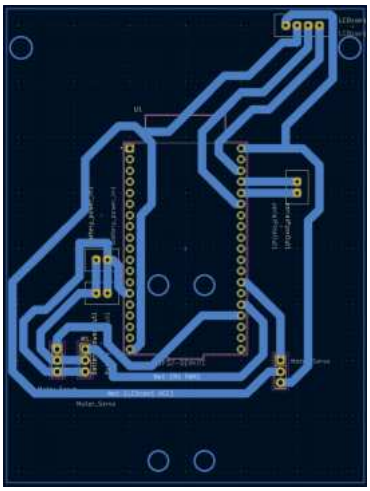
3.5 Hardware design

3.5.1 PCB

The PCB was used to neatly connect the PWM pins of the ESP32 board to the required terminal of the servo motors and to integrate the external power supply for the servos. The PCB is also equipped with an LCD display that displays each new target coordinate as well as other useful diagnostic information. The figures below (22a, 22b) show the initial design that was done in KiCad.



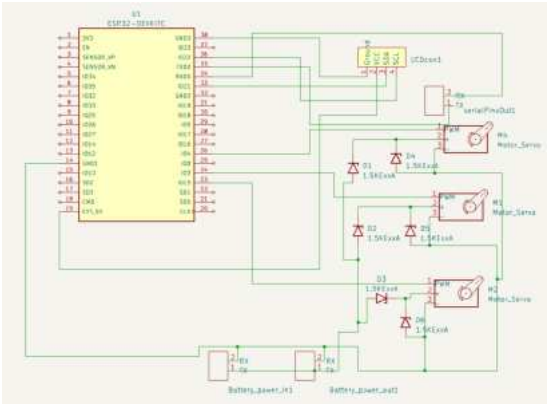
(a)
PCB schematic design



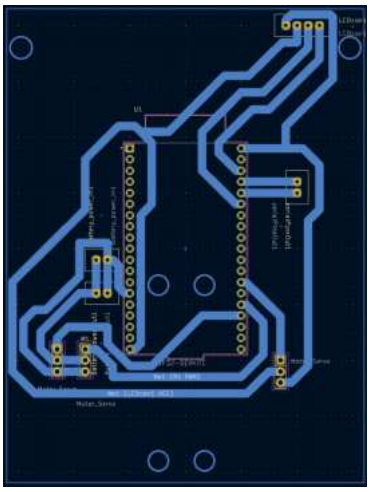
(b)
PCB circuit design

Figure 22.
PCB initial desgin

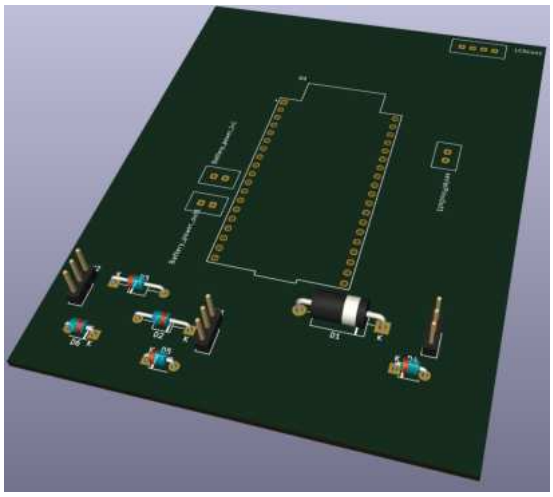
There was a fatal error in this design. The problem is called counter-electromotive force (CEMF) or back electromagnetic force (back EMF). The coils in the servos have a high inductance, the electromagnetic induction of the coils in the motors induces EMF. When EMF is introduced it drives current through the circuit. The induced current creates a new magnetic field which opposes the original magnetic field, and when the power is disconnected and the original current with the electromagnetic field is no longer present, an inverse current is induced, The motors then become generators for a brief moment causing a voltage spike and a change in polarity of the terminals. This can cause major problems. During testing, certain components would simply stop working. The LCD and one servo shared a power supply if you look at Figure 22a, it was originally thought to just be a bad LCD, until a replacement was installed which also stopped working along with the servo to control the tilt axis. The suspicion was confirmed when a new PCB was created (see Figures 23a, 23b, 23c below) with blocking diodes in place to prevent the CEMF current from returning in an unwanted direction and a freewheel diode to allow the energy to dissipate.



(a)
PCB schematic design with diodes for CEMF



(b)
PCB circuit design with diodes for CEMF



(c)
PCB 3D view with diodes for CEMF

Figure 23.
PCB final design

3.5.2 Turret

The gun turret's design was challenging due to the requirements for stability, manoeuvrability and durability. The initial attempts at the design used sheet metal as shown below in Figure 24.

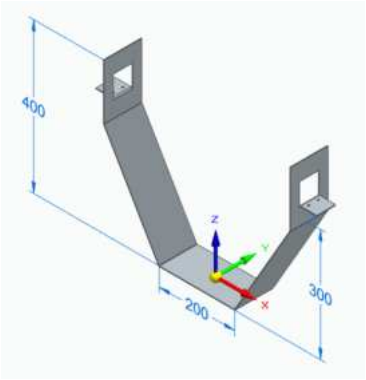


Figure 24.
Sheet metal U design

This was a conceptual idea that was never produced and would require a lot more additional parts in order to be a viable option. The simple 'U' shape supports the gun in the middle by means of a solid aluminium rod placed between the cutouts where the motors would be mounted. An additional base motor for horizontal movement would also be required.

This design would be simple and durable. However, realistically this design would encounter problems that would require a series of prototypes. A decision to use 3D printing instead was made since the prototypes are less expensive and did not require any delays that are involved with outsourcing to a third party.

The 3D print designs started with the plan of using stepper motors, but this was quickly changed to servos after finding suitable products since the stepper motors did not have the required torque and were susceptible to slipping. The stepper motors that could have met the requirements would have used the entire project budget and still required gearing. The calculations for the required torque are shown below in Algorithm (56).

$$
\begin{aligned}
I &= (1/12) * m * (l^2 + w^2)\,[\mathrm{kg \cdot m^2}] \\
\omega(\mathrm{rad/s}) &= \frac{\mathrm{degrees} * \frac{\pi}{180}}{t}\,[\mathrm{rad/s}] \\
\alpha &= \frac{(\omega - \omega_0)}{t}\,[\mathrm{rad/s^2}] \\
\tau &= I * \alpha\,\mathbf{N \cdot m}
\end{aligned}
\tag{56}
$$

In the kinematic equations shown above, m is the mass of the gun system, and l and w are length and width respectively. The moment of inertia is denoted by I and the angular acceleration is denoted by α. The gun system is around 6 kg with a length of 400 mm and a width of 190 mm. The top speed the system had to be designed for was to rotate 60° in 0.2 seconds since this was a standard top speed for most servos. The torque is then calculated as shown below in (57). Note that the time taken to reach the velocity assumes the smallest acceleration possible for the figure presented by the servos data sheet. Since the spec is 60° in 0.2 seconds it is assumed it also takes 0.2 seconds to reach this speed.

$$
\begin{aligned}
I &= (1/12) * 6 * (0.4^2 + 0.19^2) = 0.0985\ [\mathrm{kg \cdot m^2}] \\
\omega &= \frac{60 * \frac{\pi}{180}}{0.2} = 5.236\,[\mathrm{rad/s}] \\
\alpha &= \frac{(5.236 - 0)}{0.2} = 26.18\,[\mathrm{rad/s^2}] \\
\tau &= 0.0985 * 26.18 = 2.58\,\mathbf{N} \cdot \mathrm{m}
\end{aligned}
\tag{57}
$$

This translates to around 26 $kg \cdot cm$ (most servos present using this unit) since the next step up within budget was a 40 kg· cm digital servo, this is the servo that was implemented.

The turret is designed using Solid Edge and printed using a Creality Ender 3 3D printer. The full design is shown below by the two isometric drawings in Figure 25 and expanded on below.

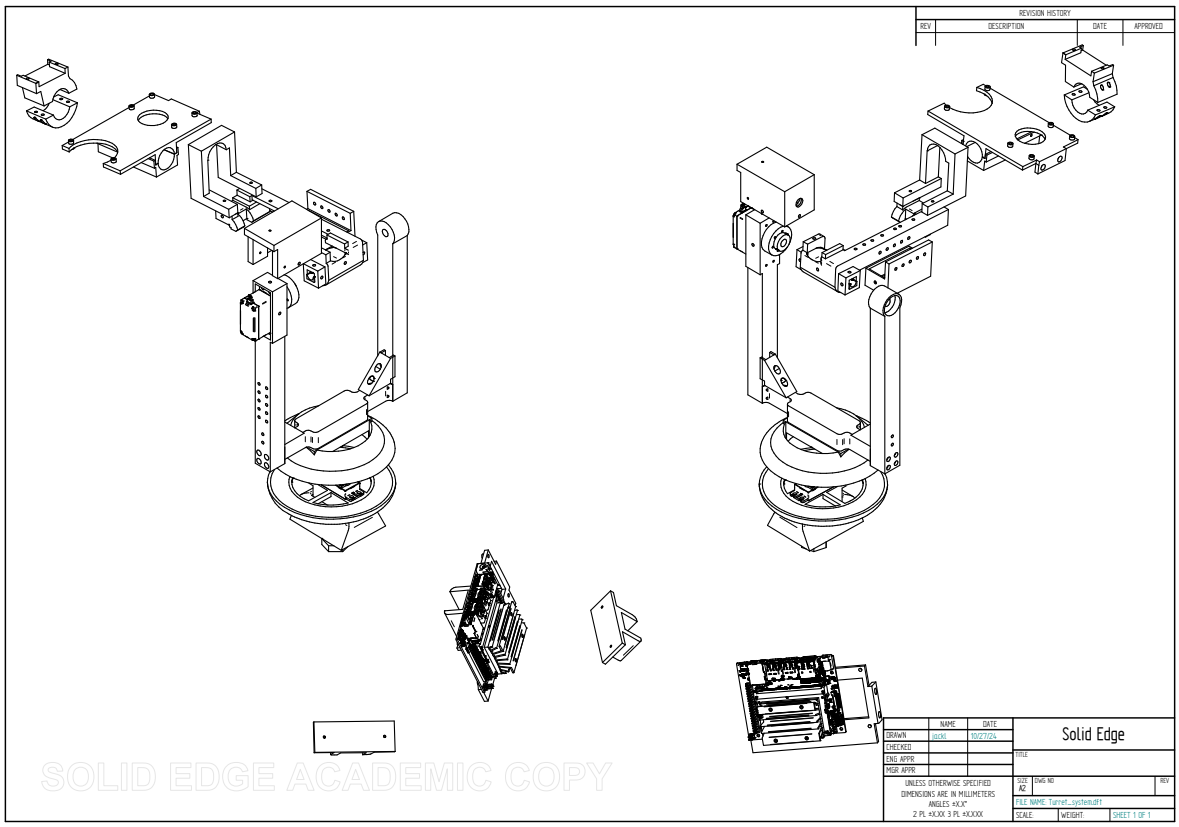
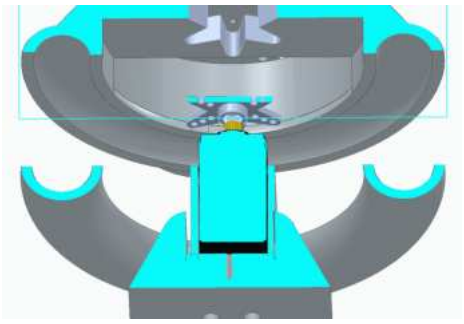
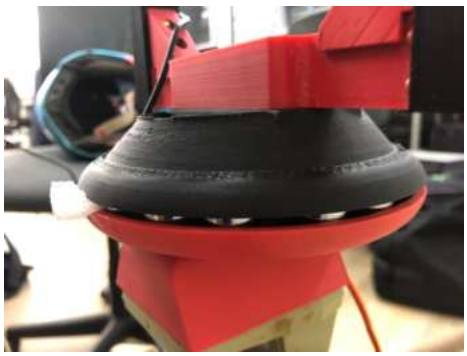


Figure 25.
Turret two view isometric drawing

The system is designed to remove all forces of the servos except for the rotational forces on the two intended axes of rotation. This is achieved by using supports and ball bearings. Notice how the bearing used for rotation in the horizontal rotational axis had to be custom-made and printed since a bearing of these dimensions could not be purchased. Below in Figures 26a, 26b one can see the design and function of this bearing.



(a)
Cross section of the base bearing of the turret



(b)
Printed turret base

Figure 26.
Turret base bearing design

The bearing is designed using two cylindrical half-tubes that have been reduced to create a vertical gap. The gap ensures all the downward force from the system sits on the inserted 9 mm steel balls instead of on the plastic, therefore, reducing friction. The final constructed turret system is shown below in Figures (27a, 27b).

(a)
Constructed turret from front left view

(b)
Constructed turret from back left view

Figure 27.
Constructed turret

3.6 Configuration

It is important to mention all the parameters that have been configured for the model so this project can be replicated and the results in the following sections can also be attained by anyone who replicates the work.

3.6.1 Training

During training and inference, it is important to keep some parameters the same since the model architecture depends on them, or because they are a chosen constant. The table below, Table 3, shows all the architecture parameters for the model during training.

Parameter	Value	Location	Description
IMAGE_SIZE	416	config.py	Input image resolution
NUM_CLASSES	4	config.py	Number of object classes
NUM_WORKERS	8	config.py	Parallel data loading workers
BATCH_SIZE	64	config.py	Training batch size
S (Scales)	[13, 26]	config.py	Output grid sizes
ANCHORS	[(81,82), (135,169), (344,319)], [(10,14), (23,27), (37,58)]	config.py	Anchor box dimensions

Table 3.
Training Architecture Parameters

The training parameters are summarized in Table 4 below. The learning rate is the rate or step size the weights are changed by as the weights are updated. The learning rate started much larger ($3e-4$) but after multiple rounds of training and slowing decreasing the learning rate, the learning rate became $0.5e-4$. The weight decay is a penalty that is added to the loss to prevent over-fitting, it simply discourages large weights in the model, expressed by equation (58) below, where $L_{original}$ is the original loss function, λ is the weight decay coefficient ($1e-5$), w represents the model's weights. The number of epochs was 100, which means the entire training dataset is passed through the network 100 times. The last training run (the one with a learning rate of $0.5e-4$ was 100 epochs). The pinned memory was to keep the data loaded for that batch in memory, since it was on the laptop which has the memory available it was set to true. The load model is used to train the weights that had already been trained with a higher learning rate. The save model was added because sometimes one needs to test parameters and didn't actually want to save the changed weights.

$$L_{total} = L_{original} + \lambda \sum_{w \in weights} w^2 \tag{58}$$

Parameter	Value	Location	Description
LEARNING_RATE	0.5e-4	config.py	Initial learning rate
WEIGHT_DECAY	1e-5	config.py	L2 regularization
NUM_EPOCHS	100	config.py	Number of training epochs
PIN_MEMORY	True	config.py	Pin memory in data loading
LOAD_MODEL	True	config.py	Load pre-trained weights
SAVE_MODEL	True	config.py	Save model checkpoints

Table 4.
Training Parameters

The training detection hyperparameters are shown below in Table 5. The confidence threshold is used to filter out all predictions that are below 62% confidence. The confidence threshold started at 0.05 so almost all predictions were evaluated. This takes a long time to process so the confidence threshold was slowly increased during training until only the predictions with a confidence of 0.62 were evaluated. The MAP IoU threshold is the IoU threshold that is used when calculating the mean average precision. The value 0.5 is a standard. The NMS IoU threshold was set to ensure only one bonding box per object was displayed. The Bachelor threshold was used, so any prediction of a class "Bachelor" and a confidence of above 0.5 is also evaluated.

Parameter	Value	Location	Description
CONF_THRESHOLD	0.62	config.py	Confidence threshold
MAP_IOU_THRESH	0.5	config.py	IoU threshold for mAP
NMS_IOU_THRESH	0.25	config.py	NMS IoU threshold
BACHELOR_THRESHOLD	0.5	runYolo.py	Bachelor class threshold

Table 5.
Training Detection Parameters

It is important to note that during training the same images/labels were never sent through the network. The data is augmented slightly to ensure the model generalises. This means each time the images and label could be rotated slightly, the colour of the image is altered slightly, or the scale is increased. Table 6 below summarises all of these augmentations.

Parameter	Value	Location	Description
scale	1.1	config.py	Scale factor
brightness	0.6	config.py	Brightness range
contrast	0.6	config.py	Contrast range
saturation	0.6	config.py	Saturation range
hue	0.6	config.py	Hue range
rotate_limit	20	config.py	Max rotation degrees
shear	15	config.py	Shear limit
p_horizontal_flip	0.5	config.py	Horizontal flip prob.
p_blur	0.1	config.py	Blur probability
p_clahe	0.1	config.py	CLAHE probability

Table 6.
Data Augmentation Parameters

3.6.2 Inference

The model parameters are slightly different for running the inference as shown in Table 7. The image size is the same since the model architecture is designed for this image size. The number of classes is the same since there are four classes. The number of workers is only 4 since the Jetson Nano CPU only has four cores, not eight. The batch size is one since the number of images getting passed through is one at a time from the video stream of the web camera. The learning rate is still used during the running statistic calculation but is set to 1e-4.

Parameter	Value	Location	Description
IMAGE_SIZE	416	config.py	Input image dimensions
NUM_CLASSES	4	config.py	Number of classes
NUM_WORKERS	4	config.py	Data loading workers
BATCH_SIZE	1	config.py	Inference batch size
LEARNING_RATE	1e-4	config.py	Learning rate

Table 7.
Inference Model Parameters

The detection parameters are also altered for inference as shown below in Table 8. The confidence threshold is increased to 0.66 since only predictions with high enough confidence should be acted upon. The MAP IoU threshold remains 0.5. The NMS IoU threshold is lowered to 0.12 since boxes that have the same class as the box being processed should only be kept if they have an IoU of less than 0.12, this ensures only one box per object is presented after NMS.

Parameter	Value	Location	Description
CONF_THRESHOLD	0.66	config.py	Confidence threshold
MAP_IOU_THRESH	0.5	config.py	mAP IoU threshold
NMS_IOU_THRESH	0.12	config.py	NMS IoU threshold

Table 8.
Inference Detection Thresholds

Finally, the configuration of the serial communications is as shown in Table **??**. The baud rate is set on the Jetson Nano and ESP32 to be 19200. The ports are the port that the web camera is attached to on the Jetson Nano on. No parity bits are used and there is one stop bit.

Parameter	Value	Location	Description
BAUD_RATE	19200	runYolo.py	Communication speed
PORT	'/dev/ttyUSB0'	runYolo.py	Serial port
PARITY	NONE	runYolo.py	Parity setting
STOP_BITS	ONE	runYolo.py	Stop bits setting

Table 9.
Inference Serial Communication Parameters

4. Results

4.1 Summary of results achieved

Intended outcome	Actual outcome	Location in report
Core mission requirements and specifications		
The system must identify the Bachelor 75% of the time in an area of 30m (width) by 20m (depth) and should never incorrectly identify a 'friendly' person as the Bachelor.	The system is achieves (90-100% accuracy) up to the 15 m marker. Further than 15 m the confidence threshold needs to be lowered to get consistent results. Lowering the confidence threshold was not a problem in the labs, but in the Engineering 3 foyer which has much harsher lighting conditions it caused many false positives for all classes besides the Bachelor.	Test 1 in 4.2
The system must identify a human among other objects with a true positive rate (TPR) of 95%.	The system achieves a human/person TPR of 98.3%	Test 2 in 4.2
The system must constantly update the set point of the PID to follow the 'unfriendly' target. The system must have up to a 0.65 m overshoot.	The PID controller achieves 6.03% overshoot and under the conditions needed for a 0.65 m overshoot (at 20 m range and around 25° bearing) the system would overshoot by 0.65 m if no new set points are given.	Test 3 in 4.2 and 3.4.8
The system must be able to identify 5 people in one frame.	The system can identify five objects of any of the four classes in a single frame but this does affect the FPS.	Test 4 in 4.2
The system must be able to keep the gun aimed at a moving 'unfriendly' at 5 frames/sec.	With a single Bachelor being tracked the frame rates are recorded between 5.16 and 5.63 FPS.	Test 3 in 4.2 and 3.4.8
At a distance of 20 meters perpendicular to the camera lens' face, the camera should be able to view 30 meters in width and the turret must be able to place the centre of its scope in any position in that range from starting position.	The diagonal field of view required to see 30 m width at 20 m depth is 75°, this field of view creates a fish-eye effect and would negatively impact the PID tracking so an automated searching feature was added to the system which searches across 180° back and forth when no object is detected to still be able to use a camera with 60° diagonal field of view and be able to detect objects in a larger area than required.	Test 3 in 4.2 and 3.4.8
Field condition requirements and specifications		
Ignore background features and don't detect them as the Bachelor.	Some background features depending on the environment do get detected as person, car, or dog but never as the Bachelor. This is very minimal when the confidence threshold is at 0.66 but it is a trade-off with the distance at which objects are detected.	Test 2 in 4.2

Table 10.
Summary of results achieved.

4.2 Qualification tests

Qualification test 1: Confidently distinguish the Bachelor from the 'friendly' person

Objectives of the test or experiment

The objective of this test is to prove that the system can correctly distinguish between a person and a bachelor class. In an area of 20 m x 30 m, the system should be able to detect the person or Bachelor at least 75% of the time and should never identify a 'friendly' person class object as an object of the bachelor class.

Equipment used

- The Jetson Nano processing platform.
- The Logitec C310 web camera.

- An ASUS G14 laptop.
- The NoMachine remote desktop service software.
- One Bachelor to enter the system.
- One willing volunteer to be the person object.
- A portable power supply for the system.

Test setup and experimental parameters

The computer vision inference script was run without the control subsystem connected and pointed in an area of 20 m in length. The hyperparameters of the computer vision model were set as shown in Table 11 below.

Parameter	Value	Location	Description
IMAGE_SIZE	416	config.py	Input image dimensions
NUM_CLASSES	4	config.py	Number of classes
NUM_WORKERS	4	config.py	Data loading workers
BATCH_SIZE	1	config.py	Inference batch size
CONF_THRESHOLD	0.66	config.py	Confidence threshold
MAP_IOU_THRESH	0.5	config.py	mAP IoU threshold
NMS_IOU_THRESH	0.12	config.py	NMS IoU threshold

Table 11.
Inference Model Parameters.

Steps followed in the test or experiment

- The camera is mounted on the gun which is mounted on the turret, and the camera is connected to the Jetson Nano.
- Mark the 5 m, 10 m, 15 m, and 20 m distances on the floor.
- The power to the turret is disconnected to ensure it doesn't move.
- Set up the network for both the Jetson Nano and laptop to connect to.
- Once both devices are connected to the same network, use NoMachine to set up a remote desktop.
- Open the project folder and run the project using the 'sudo python 3' command.
- Once the outputs start to display, ensure the camera has a clear line of sight of the testing area.
- Start screen recording the output on the laptop's remote desktop view.
- Dressed as a person (someone not wearing the blue suit and helmet or someone in the orange suit) class enter the camera's field of view and monitor the output on the laptop.

- Slowly move to the 5 m, then the 10 m, then the 15 m then the 20 m mark.
- Return back to the starting position.
- Repeat the process as the Bachelor (dressed in a blue suit and helmet).

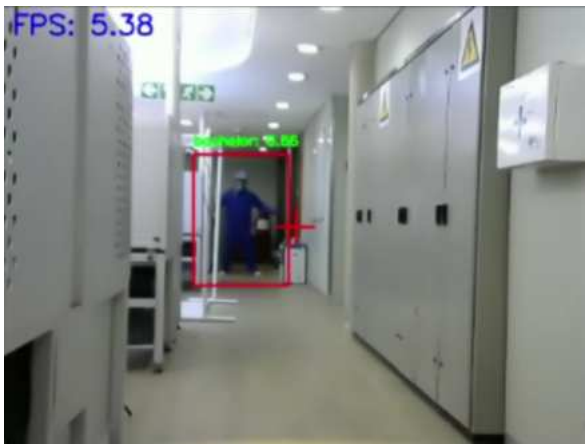
To ensure the system works with both classes at the same time, a colleague dressed as a person entered the camera's field of view at the same time as the Bachelor.

Once the experiment had been completed for each class. The recordings were reviewed and noted at which distances the objects were detected and what class they were detected as.

Results or measurements

(a)
System on Jetson Nano detecting bachelor at 5 m

(b)
System on Jetson Nano detecting bachelor at 10 m



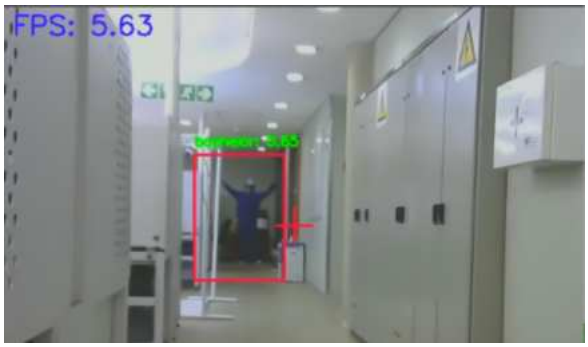
(c)
System on Jetson Nano detecting bachelor at 15 m example 1

(d)
System on Jetson Nano detecting bachelor at 15 m example 2

Figure 28.
Distance tests

Figure 29.
System on Jetson Nano detecting bachelor at 19 m confidence 0.66

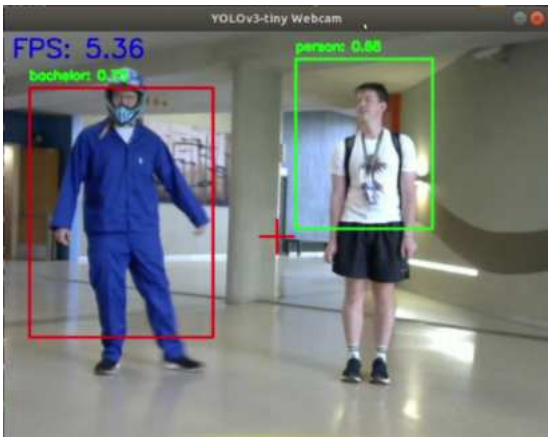
(a)
System on Jetson Nano detecting bachelor at 19 m confidence 0.64 example 1



(b)
System on Jetson Nano detecting bachelor at 19 m confidence 0.64 example 2

Figure 30.
Distance tests with lower confidence threshold

(a)
Bachelor and person in the same frame



(b)
Bachelor and person in the same frame and separated

Figure 31.
Bachelor and person in the same frame

Observations

Using the CNN predictions and then the post-processing correcting checks, there was only one time the system identified a person in the labs as a Bachelor incorrectly. He was wearing a full blue suit and for that reason, the orange safety suit was introduced (see section 6.4) and the post-processing was improved by creating smaller ROIs for checking colour characteristics. Since the improvements, no false positive Bachelor predictions have been made. From Figure 28a and Figure 28b, it is clear the system can identify the Bachelor at 5 m and 10 m in almost every single frame. Figure 28c and Figure 28d were two examples of when the system correctly identified the Bachelor but there were quite a few frames that the Bachelor was not detected. Figure 29 is an example of when the system correctly identified an object at 19 m (at the end of the lab's width), but this is an example of when it worked. For most of the time the Bachelor was present at this distance, it was not detected. To give the system a better chance at detecting the Bachelor at ≈ 20 m the confidence threshold was reduced to 0.64 and that is when Figure 30a and Figure 30b were captured. The lower confidence threshold definitely improves the detection at far distances, however, in a noisy (cluttered) environment, lowering the confidence threshold could result in many false positives for all classes.

Using the results from the distance tests, the diagram below in Figure 32 reflects the range of the system in the 20 m x 30 m area. The purple cross-hatched area is the area the Bachelor would get detected.

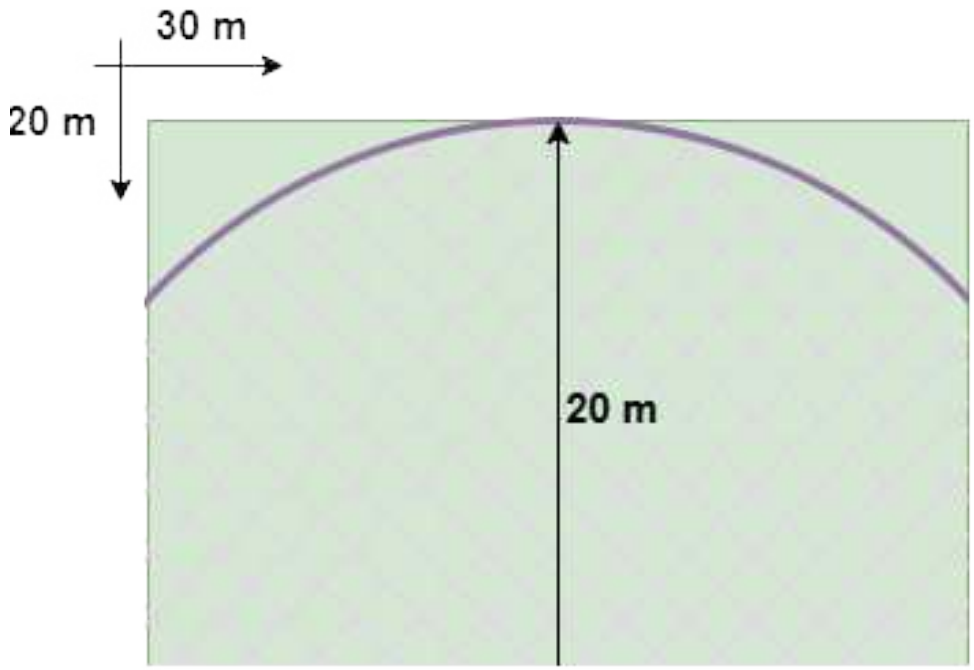
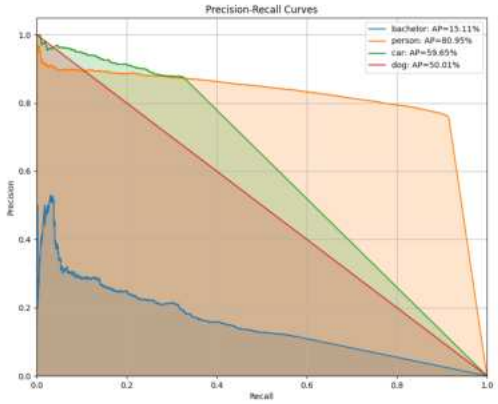


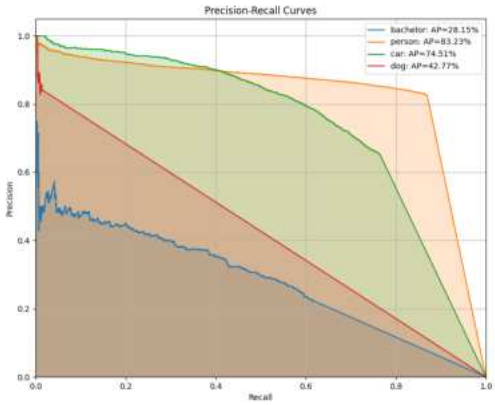
Figure 32.
System capable area

Statistical Analysis

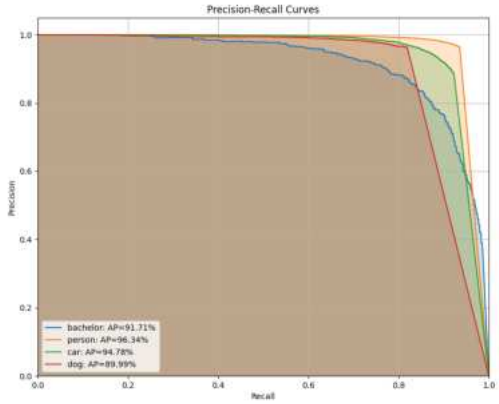
Training from epoch zero using the custom dataset and configuration as described in section 3.6. During training, the code was added to calculate the PR curves for each class at epoch numbers 10, 50, and 200. The results from epochs 10 and 50 are included to verify that the model was learning and that the PR curves were reflecting realistic results for these points in training.



(a)
The PR curve at the being of training after 10 epochs



(b)
The PR curve during training after 50 epochs



(c)
The PR curve at the end of the training after 200 epochs

Figure 33.
AP crurves

```
Class accuracy is: 96.384666%
No obj accuracy is: 98.232559%
Obj accuracy is: 51.771511%
Final evaluation:
100%|                                                                                | 14/14 [01:04<00:00,  4.58s/it]
Class accuracy is: 94.833687%
No obj accuracy is: 98.520287%
Obj accuracy is: 41.507431%
100%|                                                                                | 14/14 [01:12<00:00,  5.21s/it]
Final MAP: 0.41191715002059937
=> Saving checkpoint
```

Figure 34.
mAP result

The Average Precision (AP) scores show an impressive improvement for all classes, with the person class reaching 96.34%, the bachelor class reaching 91.7%, the car class reaching 94.78%,

and the dog class reaching 89.99%. The person class consistently performed best across all training stages. This is most likely because there were so many training samples for the person class. The bachelor class showed the greatest improvement, moving from 15.11% to 91.71%. In Figure 33a and Figure 33b the classes have steep drop-offs which suggests trade-offs were being made. In Figure 33c all the classes have a high consistent performance suggesting that the model has converged. The mean average precision result shown in Figure 34 is discussed in section 5.1.1.

Qualification test 2: Person object detection and recognition

Objectives of the test or experiment

The object objective of this experiment was to prove how well the system can identify the person class. As stated in the project proposal requirement 2, the goal was that the system must be able to distinguish people from other objects and animals. It is for this requirement that the car and dog classes were included since these are the two most likely classes to ever be in the same area as the system's area of operation. The goal is to achieve a TPR of 95% for the person class.

Equipment used

The only equipment required for this experiment is the laptop used to train the model which is the ASUS G14 equipped with an RTX 2060 GPU.

Test setup and experimental parameters

It is important to note the custom dataset that was used for training. Since the more samples one has to train a CNN the better the generalisation is for each class. Two thousand examples from the PASCAL VOC dataset were extracted that only contain "person", "car", and "dog" classes, and then combined the images and label files to form one dataset with the 500 samples taken of the Bachelor and created a train and test CSV file. The Table 12 below shows the parameters used during training when the graphs were plotted.

Parameter	Value	Location	Description
LEARNING_RATE	0.5e-4	config.py	Initial learning rate
WEIGHT_DECAY	1e-5	config.py	L2 regularization
NUM_EPOCHS	200	config.py	Number of training epochs
CONF_THRESHOLD	0.62	config.py	Confidence threshold
MAP_IOU_THRESH	0.5	config.py	IoU threshold for mAP
NMS_IOU_THRESH	0.25	config.py	NMS IoU threshold

Table 12.
Training Parameters

The training code was altered to calculated and display the confusion matrix at epochs number 10, 50, and 200.

Steps followed in the test or experiment

- Using the training version (heavily optimised with libraries and only designed for training) of the model, set the hyper-parameters as shown in Table 12.
- Run the training script on the model for 200 epochs.

- The full training takes around 12 hours for 200 epochs on the custom dataset, so train overnight.
- Retrieve the confusion matrix images generated from epochs 10, 50, and 200.
- Analyse the results and draw conclusions.

Results or measurements

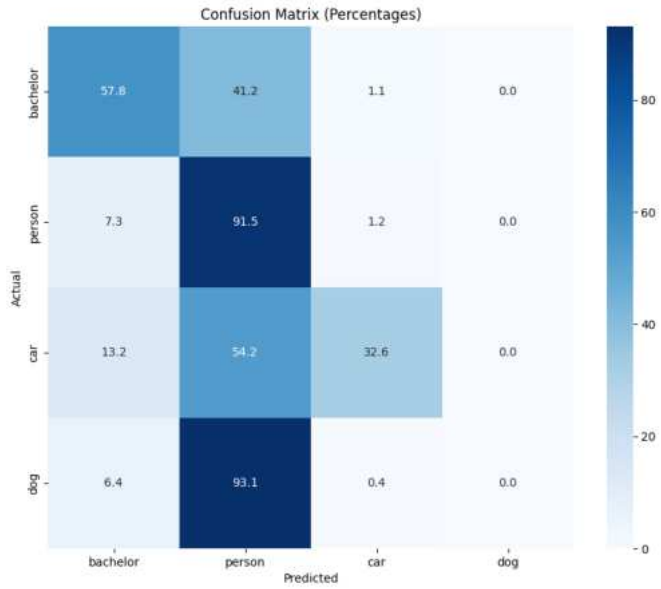


Figure 35.
Confusion matrix at the being of training after 10 epochs

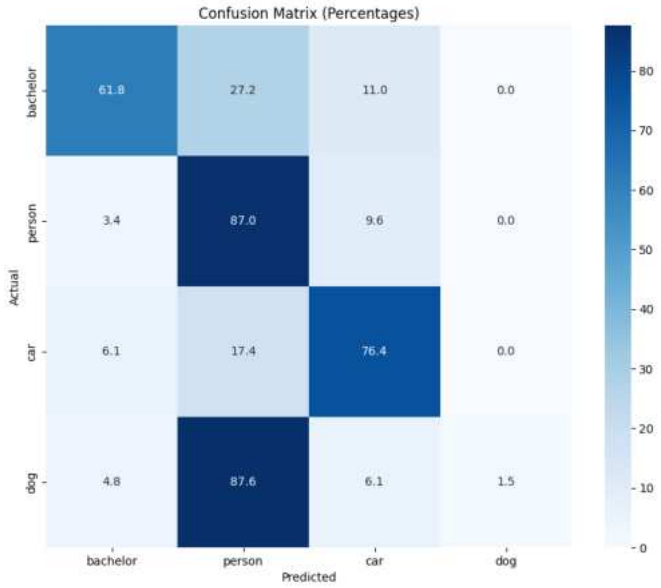


Figure 36.
Confusion matrix during training after 50 epochs

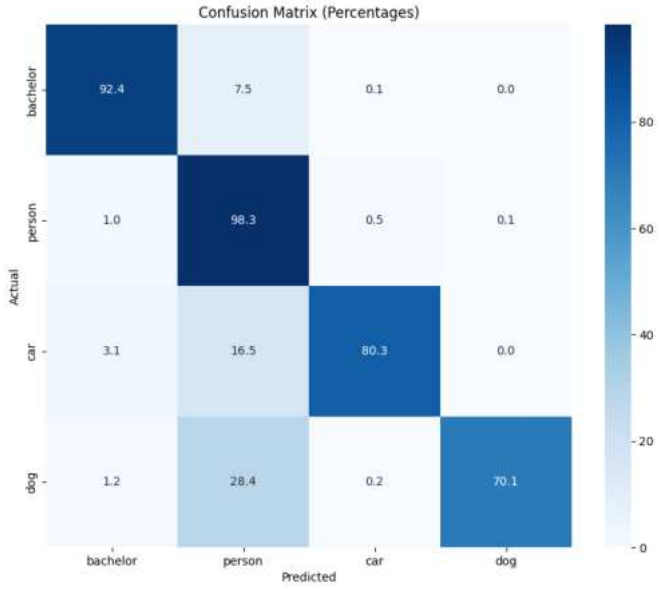


Figure 37.
Confusion matrix at the end of the training after 200 epochs

Observations

From the confusion matrices generated, one can observe significant differences in different training stages. In the first 10 epochs version of the function matrix shown by Figure 35, the person class already had a TPR of 91.5%, however, the TPR of the bachelor class was only

57%, the car class TPR was only 32.6% and the dog class TPR was 0%. The biggest problem was that 41.2% of the time the model was predicting the bachelor class objects as person class objects. The reason so many predictions for bachelor class objects were labelled person class is simple. Since the model tries to generalise what a person object is using the different layers with automatic feature extraction, one ends up describing the person shape of many colours and dimensions as a person and then to try and single out a specific person from this generalisation is like trying to define two separate classes, but one of them is actually a child/subclass of the other. The second Figure 36 created at epoch 50 shows many improvements. Although the TPR of the person class had dropped to 87% the bachelor class TPR had increased to 61.8% and the percentage of bachelor objects that were mislabelled as person objects had also dropped to 27.2%. The car class TPR had increased to 76.4% but the dog class TPR is only 1.5%. By the end of training at epoch 200 the confusion matrix generated is shown in Figure 37. After 200 epochs of training the person class TPR was 98.3% which is greater than the requirement of 95%. Besides the requirement, the TPR of the bachelor class is 92.4% and this is just from the output of the CNN and not with the additional checks implemented which are explained in section 3.4.2. One can also see the TPR of the car and dog classes have increased to 80.3% and 70.2% respectively.

Qualification test 3: The PID controller, turret, and frame rate

Objectives of the test or experiment

The PID controller test is set up to prove that requirements 3, 5, and 6 from the project proposal have been achieved. This test aimed to demonstrate the designed overshoot in the PID system in both simulation and practice as well as the turret's ability to be able to swivel across the full field of view of the camera. The design requirement and the measurable objective of the test was to achieve a 5% overshoot for the PID controller and that the turret can manoeuvre the gun to point at a Bachelor position in a 20m x 30m area. The last part of the test was to monitor the FPS while tracking the Bachelor. The objective was to achieve 5 FPS.

Equipment used

- The Jetson Nano processing platform.
- The Logitec C310 web camera.
- The ASUS G14 laptop.
- The NoMachine remote desktop service software.
- One object resembling the Bachelor.
- The gun and turret structure.
- The portable power supply for the system.
- The MATLAB simulation software.

Test setup and experimental parameters

The Jetson Nano inference version of the model is set up with the parameters as shown in Table 13 and the experiment is conducted in the large area of Engineering 3 foyer on Saturday so not many other people could interfere with the test.

Parameter	Value	Location	Description
IMAGE_SIZE	416	config.py	Input image dimensions
NUM_CLASSES	4	config.py	Number of classes
NUM_WORKERS	4	config.py	Data loading workers
BATCH_SIZE	1	config.py	Inference batch size
CONF_THRESHOLD	0.66	config.py	Confidence threshold
MAP_IOU_THRESH	0.5	config.py	mAP IoU threshold
NMS_IOU_THRESH	0.12	config.py	NMS IoU threshold

Table 13.
Inference Model Parameters

Steps followed in the test or experiment

- The camera was mounted on the gun which was mounted on the turret, and the camera was connected to the Jetson Nano.
- The power to the turret is connected to ensure it can move.
- Set up the network for both Jetson Nano and laptop to connect to.
- Once both devices were connected to the same network, the NoMachine software was used to set up a remote desktop between the devices.
- The project folder was opened and the inference script was run using the 'sudo python 3' command.
- Once the outputs started to display, the system was aligned for a clear line of sight of the testing area.
- Screen recording was started on the laptop to record the output.
- The bachelor (someone wearing the blue suit and helmet) class entered the camera's field of view and the output was monitored on the laptop.
- The Bchelor slowly moved across the radius of the area ensuring the gun followed.
- The Bachelor returned back to the starting position.
- The screen recording was analysed and results were taken.

Results or measurements

(a)
Bachelor moving out of the frame to the right

(b)
Bachelor moving out of the frame to the right and gun system following

Figure 38.
Tracking the Bachelor results 1

(a)
Bachelor moving out of the frame to the right and gun system following and overshooting

(b)
Bachelor moving out of the frame to the right and gun system following and then settling

Figure 39.
Tracking the Bachelor results 2

Observations

Although the images displayed in the results section were selected because they were good examples, and during the experiment, there were many false positive predictions due to the harsh lighting, the system never made a false positive Bachelor prediction. The system was able to detect the Bachelor consistently while moving at a walking pace in a distance of 5 m to 15 m. Moving quickly and at a distance of 15 m to 20 m the system would detect and act periodically. In Figure 38a the Bachelor walked out of the system's field of view to the right. The system

identified the Bachelor object and then using the last known direction protocol moved in the last known direction of the Bachelor to attempt to find the Bachelor again. The system found the Bachelor again in Figure 38b when the Bachelor slowed down allowing the system to catch up. The system then overshot the target due to the PID being designed for a 5% overshoot in Figure 39a. The system then settled on the Bachelor and started shooting (pulling the trigger, no actual shooting took place) in Figure 39b. For most of the experiment (and in all figures presented), the FPS stayed above 5. However, in areas where the camera was directed towards a window and direct sunlight was facing the camera and reflecting off the glossy floor, the system would make more false positive predictions and the frame rate would drop a few times to as low as 4.7.

Statistical Analysis

The PID simulation was covered in the design Section 3.4.8. The final results of the design are shown below in Figure 40 and Figure 41.

Figure 40.
PID Simulink simulation output after tuning

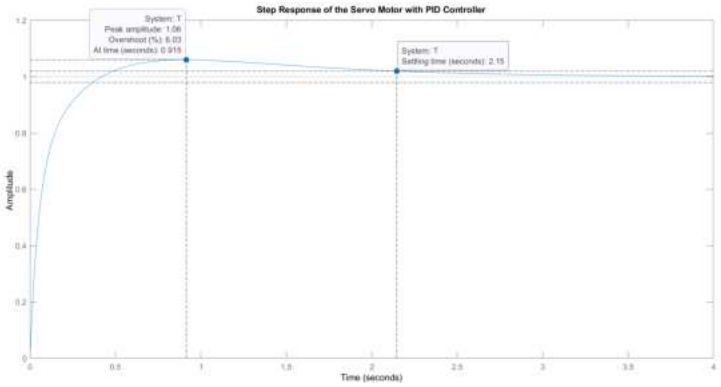


Figure 41.
PID simulation output after tuning

The PID response shown in Figure 41 shows an overshoot of 6.03%, which at 20 m away with a target 25° to the right could result in the turret moving 26.5075°. At 20 m away moving 25° results in a 9.326 m of lateral movement perpendicular to the system for the Bachelor moving in a straight line. At 20 m away moving 26.5075° results in a 9.9749 m of lateral movement perpendicular to the system for the Bachelor moving in a straight line. This means the overshoot would be 0.6487 m which is rounded to 0.65 m. This is an example chosen to reflect how the 0.65 m overshoot could be attained but it is just an example. In practice, there would be multiple

new set point inputs between the time the 25° measurement was calculated and the time the gun got the position of the Bachelor and the varying change in the set points would have an effect on the outcome and overshoot. The example shows that the system would overshoot correctly (by 0.65 m) in this example.

Qualification test 4: Multiple object tracking

Objectives of the test or experiment

The objective of the test is to show the ability of the system to satisfy requirement 4 of the project proposal which is to be able to detect five person objects in a single frame.

Equipment used

- The Jetson Nano processing platform.
- The Logitec C310 web camera
- The ASUS G14 laptop.
- The NoMachine remote desktop service software.
- Five objects resembling the person class.
- The gun and turret structure.
- The portable power supply.

Test setup and experimental parameters

The Jetson Nano version inference script version of the model is loaded onto the Jetson Nano and the parameters shown in Table 14 below are used.

Parameter	Value	Location	Description
IMAGE_SIZE	416	config.py	Input image dimensions
NUM_CLASSES	4	config.py	Number of classes
NUM_WORKERS	4	config.py	Data loading workers
BATCH_SIZE	1	config.py	Inference batch size
CONF_THRESHOLD	0.66	config.py	Confidence threshold
MAP_IOU_THRESH	0.5	config.py	mAP IoU threshold
NMS_IOU_THRESH	0.12	config.py	NMS IoU threshold

Table 14.
Inference Model Parameters

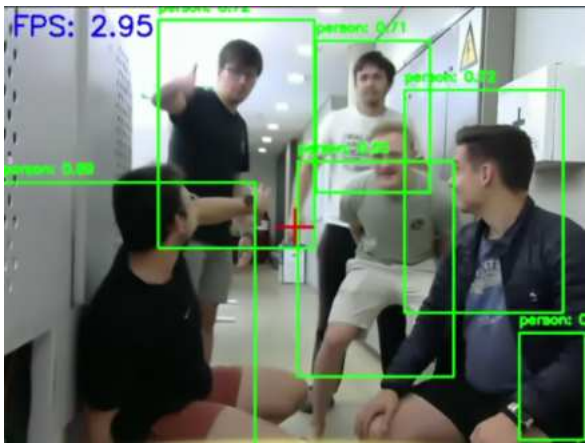
An area of sufficient size for five people was selected and only the computer vision subsystem was enabled. The control subsystem was disabled.

Steps followed in the test or experiment

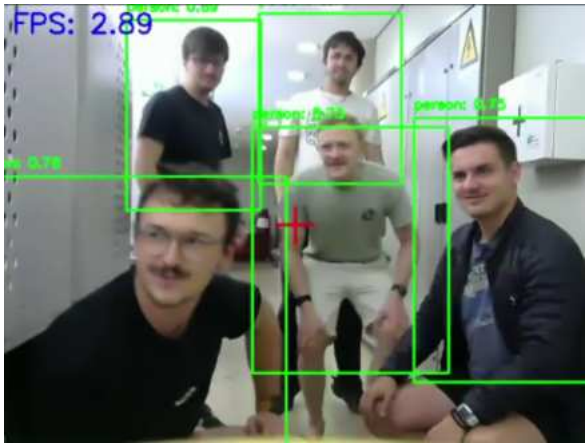
The camera was attached to the gun and connected to the Jetson Nano which were both mounted on the turret system. The system was set in an area large enough for five people and pointed

in a direction to face the people. The Jetson Nano and laptop were both connected to the same wireless hotspot to allow remote desktop connection using NoMachine. The output was recorded on the laptop which had the remote desktop to the Jetson Nano setup. The inference script was run and the five people moved within the camera's field of view. Example frames from the output recoding are shown below in Figures 42a, 42b, and 42c.

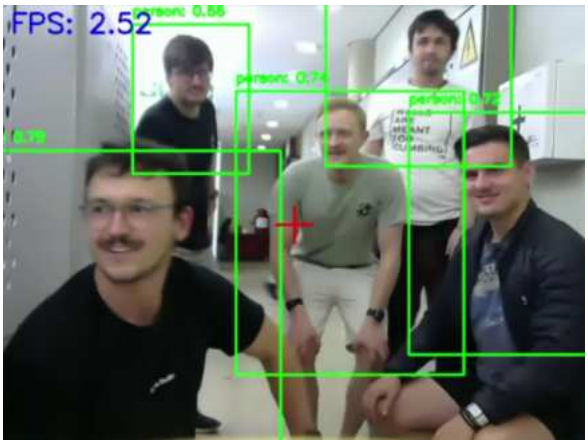
Results or measurements



(a)
System on Jetson nano tracking 5 objects example 1



(b)
System on Jetson nano tracking 5 objects example 2



(c)
System on Jetson nano tracking 5 objects example 3

Figure 42.
Multiple object tracking.

Observations

During the test, the system successfully managed to constantly detect the five people in the frame. It was noted that due to the low NMS IoU threshold, if two people got close together only one large bonding box would appear for both of them. The figures above were images from the screen recording where everyone was separated far enough apart to have their own bonding box. If the test was to be repeated, the test would be conducted in a larger area and people could be

spread apart further. Note the frame rate in the top left corner of each image. The frame rate dropped significantly when multiple objects were detected in each frame. The system was able to track five objects at once and passed the test.

5. Discussion

5.1 Critical evaluation of the design

5.1.1 Interpretation of results

As a complete system, the results show good performance, however, the system is not perfect. In qualification test 1 in section 4.2 the real-time experiment showed that the Bachelor and other objects can be detected far greater than 75% of the time while within a range of 15 m or less. However, when the objects moved past the 18 m mark the confidence threshold had to be lowered in order to get consistent bounding boxes for the objects. After reviewing the recording of the experiment there are several interesting details to acknowledge. The reason for the Bachelor not getting detected at the far distance could be because at that distance (given that the system resized the image to 416 x 416 pixels), there are very few pixels representing the Bachelor but this is what the anchor boxes and scale predictions were designed for. Upon further review, it was noticed that when the confidence threshold was dropped, there were a few very small (small bounding boxes) false positive predictions made on small objects like pencil cases that slightly resembled the human figure. This showed that the system did create the predictions for the very small objects. This means that either the training data (images of the Bachelor suit) did not have enough samples of the Bachelor at very far distances, or the lighting conditions in the test area towards the far end were too dim (see Figure 29) and the automatic feature extraction could not detect the objects.

The qualification test 1 statistical analyses reflects an almost perfect system. These PR curve results indicate reflect a system that performs better than the system performs in the real-time tests. After 200 epochs the curves are very close to perfect, see Figure 33c. Comparing it to the research papers [1] and [6] it seemed as though the training and testing datasets were too similar and this results in the model performing so well in the tests. The solution to not having realistic results was to get another metric. The mAP metric was calculated with no further training on the model than what had already been done and the following results in Figure 43 were obtained.

Figure 43.
mAP result

Comparing the mAP score of 41.19% to the mAP scores in published work which was 52.7% and 51.5% and remembering that these were larger models, this metric seems more reliable than the PR curves. Although the model is not as precise as the larger models made in the research papers, it does outperform the mAP scores of the real-time detectors, 100Hz DPM (mAP of

16%), and 30Hz DPM (mAP of 26.1%). This makes the implemented computer vision system a suitable solution.

The rest of the results were within reason to believe, and no new metrics were needed. The computer vision model is not better than any previously created models and does not match the performance of the latest implementations that have been published. However, the system was designed for a limited recourse processing platform, and it is tested on the limited resource processing platform (the Jetson Nano). Comparing it to models that were designed for different platforms, and tested on higher resource platforms does not fairly reflect the achievements this system has made. The system averages above five FPS while tracking a single object. The system controls a paintball gun that then shoots the target. The Hardware although not incredibly complex compared to the developed software, does exactly what it was designed to do. The PCB performs its tasks of powering all the components, enabling communication between components, and has surge protection after installing the diodes for the CEMF handling. The PID controller meets the overshoot requirement and slows the servo's motion making the turret's movements smoother.

5.1.2 Critical evaluation

The pivotal choices in the project (in order of importance) were to use a YOLO-like CNN structure, to separate the system into two subsystems (computer vision and control), dataset design, post-processing the output with verification, and revisiting the PCB design.

The choice to to make a YOLO-like CNN computer vision system came from research and advice from the supervisor who believed the 5 FPS spec would be hard to achieve. Knowing this, literature was researched for the fastest computer vision solution that could generalise and be robust enough to work in different light conditions and environments. Since the system had to work in different locations and lighting, the option for a solution such as template matching was no longer viable, and an NN approach was needed. Not using the YOLO-like model would have been a mistake and it was a good design choice to simply the model in order to achieve the speed requirements.

The decision to separate the subsystems was driven by concern. Controlling motors from the Jetson Nano is known to be a risk since the motors can damage the processing platform under certain conditions. The PCB was redesigned because of CEMF which had destroyed an LCD and servo, but could also have damaged the processing platform if the system was not split into the subsystems.

The dataset design was a difficult process, as training the network takes a long time and the quality of the data used determines the quality of the weights that get produced. In the end, a few Python scripts were made which searched through the Pascal VOC dataset to find good images with their labels and then combined them with the photos and labels created of the Bachelor to make a dataset to train and test the network. The simple principle is Garbage in, garbage out (GIGO).

The post-processing verification of predictions was the final key to the software side. It ensured that no person would be falsely predicted as the Bachelor making the system safe and reliable.

The final change was the PCB, it was essential. Without dealing with the CEMF it would be unclear as to what would get damaged next or when.

The only choice could be reconsidered if the system were to be implemented again would be the choice to use servos. The choice of servos was forced by my requirement for torque and speed with a budget. The servos were R 120 each but stepper motors with the same capabilities would have been R 1500 each. The system could be improved if the investment for stepper motors was a possibility and then use a reliable gearing mechanism.

5.1.3 Unsolved problems

The system works well in lit environments, but in very low light conditions the camera can not see anything. This is a hardware constraint of the camera. If one was to have the funding to use a camera capable of thermal vision this system could work in complete darkness and be used for many more applications such as nocturnal pest control on farms.

The turret structure is suitable for the means of the project but for a real-world application, paying the money for a metal structure which would be a much more durable solution and would be a good decision.

The distance problem could be fixed by using a smarter camera with adjustable zoom. The coordinates would be adjusted based on the zoom before being sent to the PID controller. This could increase the range a lot depending on the camera.

5.1.4 Strong points of the design

The architecture of the CNN was well thought out, ensuring the down-sampling convolution layers got to the correct dimensions for the scale predictions and the size of the network meant the five FPS requirement was met.

The support for the servos was a great design and executed well. The design allowed the servos to control the gun freely in the intended rotational axis while moving all the other forces onto the bearing and supports.

The PID controller works as intended. The design was kept simple and edge cases that caused failures were examined and dealt with by creating alternative solutions to perform the function if one function fails. The PID has a very fast refresh rate and is easy to tune.

5.1.5 Expected failure conditions

The system should perform as expected in good lighting, which means no glare directly into the lens of the camera and no low light conditions. If there is too little light for the camera to see the object, the system will not predict the object and the system will not act on any predictions.

The system will shoot a Bachelor when walking slowly. However, when a Bachelor is running they will be able to evade the paintballs until they slow down and allow the control system to "catch up". The system will find the Bachelor in an area, but if the Bachelor is in a continuous run the entire time, it is unlikely the system will be able to shoot them.

The system is sturdy enough to support its own movements. However, if an object prevents the turret from moving the servos could be damaged and the system will no longer be able to move.

5.2 Considerations in the design

5.2.1 Ergonomics

The system is designed to be transported to the site where it will be used. The system simply pulls apart into two sections, the base tripod, and the gun. The base tripod section has the tripod, Jetson Nano, and both power supplies. The gun part has the top section of the turret, gun, camera, and paintballs. The gun top section then folds after loosening two bolts, and the system can be stored for transport. The design of the top section allows access to the gun grip, allowing the user to pick up the section just as they would the normal gun.

The User Interface (UI) is designed for very simple use with a run and train button. The run button runs the inference scripts to start targeting the Bachelor object and shooting while displaying what the barrel is pointed at. The training is used if the system is struggling to detect the Bachelor in a certain environment. It takes 50 images of the Bachelor and presents a labelling UI to the user to quickly click and drag the bounding boxes which then get combined with a small training dataset which trains the CNN for a few epochs and updates the weights.

5.2.2 Health and safety

The added post-processing verification protocols apply a series of colour masking tests on different ROIs of the bounding box if the prediction is of the bachelor class, and if any of the tests fail, the class is changed to person. For added safety during testing, the plastic wedge is placed in the barrel of the gun to ensure nothing can be propelled out of the gun.

The orange suit was included as an option for a person who closely resembles the Bachelor. If a person needed to enter the systems area of operation but their clothing closely resembled the Bachelor's suit (blue long pants and blue long-sleeve shirt), the person could wear an orange suit to ensure they would not get identified as the Bachelor. This is explained in section 6.4.

5.2.3 Environmental impact

The environmental risk of the system, especially if used for something like farm pest control is the 3D-printed plastic. The plastic is not very durable and will eventually fail. If the structure fails, then plastic pieces will litter the area where the system is situated, polluting the environment. If animals are within the area, they could eat the plastic and this could cause various health issues for the animals.

The system also has electronics that are not weather-resistant. If the system was placed outdoors in harsh environments, there is the possibility of causing damage to batteries, electrical fires and more. The damaged batteries must be disposed of properly, and fires caused by battery damage could cause major environmental and ecosystem damage.

5.2.4 Social and legal impact

This system has serious social and legal impacts. It is illegal to create "booby traps" or deterrents that can cause serious harm. If the system is set up for pest control on a farm and the system shoots a worker there could be legal repercussions.

The social impact is dependent on the application. If set up as described in the project for a paintball game, the social implications are minimal. If the system is used for home protection it could cause social unrest and some people may be concerned by the idea of a computer vision gun that can target people based on appearance.

5.2.5 Ethics clearance

The project did not require any ethical clearance since no paintballs were ever fired and the project could be seen as a proof of concept design.

6. Conclusion

6.1 Summary of the work completed

This report describes work carried out on a computer vision auto-following paintball gun system, with the objective of recognising a specific person and shooting them. A literature survey was completed on modern computer vision algorithm design. The hardware and software for the paintball gun system were designed from first principles. The system used an existing development board with a dedicated GPU for high-speed graphics processing. The system made use of an existing camera, servos and microcontroller. All other hardware was designed and implemented from first principles. A MATLAB program program was developed to simulate the PID control system, as well as Python code for the processing platform, and a variant of C for the microcontroller which resides on the PCD that was designed and implemented. The system was implemented and several field tests were conducted. The tests were conducted in the Engineering 3 project labs on the University of Pretoria campus. The main results are shown in Figure 37 in section 4.2 and test 1 in section 4.2.

6.2 Summary of the observations and findings

The system correctly distinguished between the person and Bachelor classes consistently within a 15 m range, allowing tracking by detection. Objects further than 15 m away were inconstantly detected, but the system still classified the object class correctly if detected. When a Bachelor object was detected, the system could follow the object keeping the barrel of the gun pointed towards the Bachelor object and shoot it if the object was moving at a slow to medium-slow pace. Fast-moving objects are lost by the tracking system since they move out of frame. However, the system can relocate them and continue tracking them because it records the last known direction of the Bachelor and will start searching for the Bachelor by moving in that direction.

An important discovery was that the CNN struggled to create the generalisation of the person class while trying to classify the Bachelor (which is a subclass of the person class), as a separate class. Creating a custom weighted loss function to ensure that the bachelor class was prioritised just as much as the person class during training, despite the person class having many more training samples than the bachelor class. This was further improved by adding one last level of feature extraction outside of the CNN structure and used to verify predictions.

6.3 Contribution

The CNN and YOLO's unique architecture had to be learned. Implementing a CNN is not covered in any undergraduate modules and all the mathematics has to be learnt. The biggest challenge was to learn what to do when it does not work. Unlike usual software applications, when one can set a breakpoint and debug, to see where the values do not make sense, is a more complex process with a CNN, since the statical output from each layer such as the variance or mean in the weights of the filter or nodes can look correct they don't provide a precise problem point.

Image processing and colour theory have to be implemented which is not covered in any undergraduate modules. Learning which formats are best for certain applications such as colour masking and retrieving ROIs.

The PCB design and implementation were completed on KiCad a new software design tool. The PCB was constructed using copper board, an inkjet printer, a torn piece of magazine paper, a hot iron and ferric chloride, a process which is not taught in any undergraduate program.

The hardware was designed and simulated using Solid Edge. The software is not taught in any undergraduate module. This software had to be learnt and mastered in order to complete the turret.

The code was mostly developed by the student with little to no reliance on the existing libraries (except for the training files). Only code that relates to hardware and GPU interfacing and basic maths functions such as matrix multiplication was taken directly from existing libraries. For training the model, Pytorch libraries were used. The model which was developed to run on the Jetson Nano processing platform used tensors to allow utilisation of the GPU and Compute Unified Device Architecture (CUDA).

6.4 Future work

The turret system, although reliable, could use a better means of control such as using stepper motors with a capstan drive instead of gears, as 3D-printed gears can fail.

The system does meet the speed requirement of five FPS however, with time the project could get converted to C++ which should improve speed a bit.

The camera is fine for the requirements of this project but implementing a high-performance camera with dynamic zoom and logic to adjust the aim based on the zoom could improve the system a lot. However, this camera equipment is expensive. The other alternative for improving the camera would be to use a thermal camera, but this would require changing the requirement to target a class of a specific shape, not a specific shape wearing certain clothing.

7. References

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[2] Y. Zhang and Z. Qiang, "Real-time person detection and recognition based on yolov3 and deep sort," in *Proceedings of the IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 1234–1238.

[3] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6469–6477. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2017/papers/Hosang_Learning_Non-Maximum_Suppression_CVPR_2017_paper.pdf

[4] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 658–666, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8953982

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer, 2016, pp. 21–37.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016, pp. 779–788.

[8] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, 2017. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=8308186

[9] Z. Zheng, R. Ye, P. Wang, D. Ren, J. Li, and M. Wang, "ICIoU: Improved loss based on complete intersection over union for bounding box regression," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–10, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9497076

[10] K. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 4, pp. 559–576, 2005.

Appendix

Datasets

The datasets consist of two folders, images, and labels. In the images folder is all the images which have been named (with a Python script) from ’000000.jpg’ upwards. In the labels folder is the same amount of .txt files which have the same name as their corresponding image from the image folder along with one extra file (classes.txt). The Figures 44, 45 below show the contents of the images and labels folders respectively.

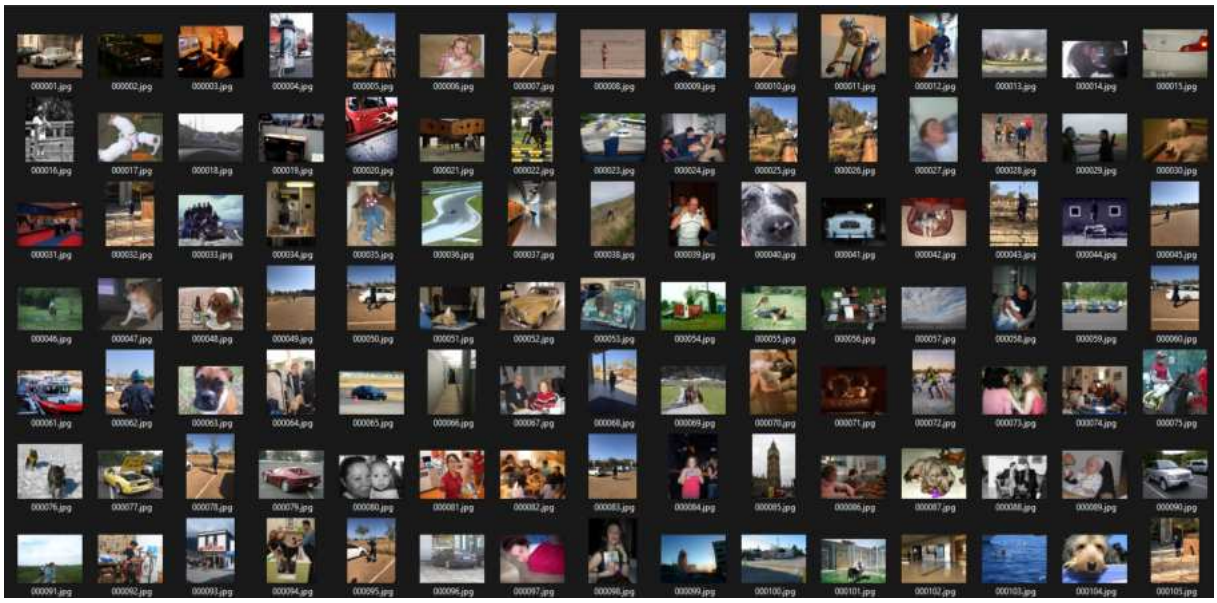


Figure 44.
Screen shot of the images folder for a dataset

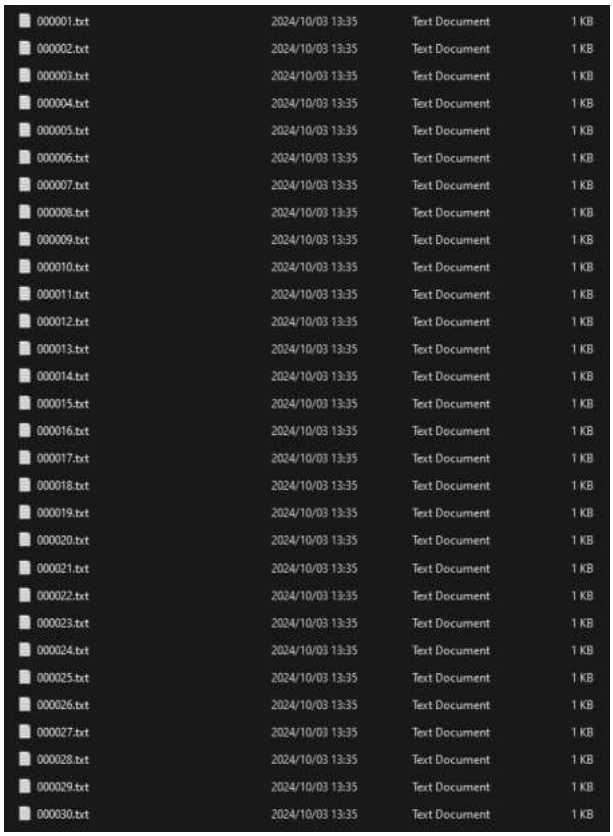


Figure 45.
Screen shot of the labels folder for a dataset

Each of the labels '.txt. files contain the information for the bonding boxes for the image it is paired with. Figure 46 below is an example of one of the label text files.

```
1 0.2852852852852853 0.804 0.18018018018018017 0.38
1 0.14714714714714713 0.679 0.04204204204204204 0.074
2 0.6381381381381381 0.672 0.11711711711711711 0.056
2 0.055555555555555555 0.683 0.1111111111111111 0.074
2 0.8303303303303303 0.639 0.27927927927927926 0.106
1 0.44144144144144143 0.605 0.21621621621621623 0.194
```

Figure 46.
Screen shot of a label text file

The first number of each line is the class of the object that the bonding box has covered. This is an index number from 0 to three referring to the list ['bachelor', 'person', 'car', 'dog']. The second number of each line is the x-axis coordinate of the centre of the box, and the third number is the y-axis coordinate. The last two values of each line are the width and height of the bounding box respectively.

Safety

The orange suit was a safety measure that could be used in the event that a person needed to enter the system's area of operation but their appearance closely resembled the Bachelor. To give an example, if a paintball arena monitor needed to walk through the area where the gun was operating but they were wearing blue jeans and a blue long-leave shit, which looks similar to the blue overalls the Bachelor wears. The monitor would put on an orange item of clothing. Wearing orange (the complementary colour to blue) ensures that the system would not predict the person as a bachelor class, and if there is a false prediction, the verification test during the post-processing would change the class to person and the person would not be targeted or shot. Figure 47 below shows the orange suit being detected by the system as a person class.

Figure 47.
Screen shot of a person in the orange suit being detected