SERVOSENTRY

Design and Implementation of a Hybrid Locomotion Robot Using Computer Vision for Autonomous Security

ECE 4600: G08

Members

Bryce Cadieux
Fola Adegoke
Noah Stieler
Peter Eshikena
Kazi Al Nahian
Seth Moule

Advisors

Colin Gilmore
Ian Jeffrey

March 14, 2025

Abstract

ServoSentry is a cutting-edge, mobile, computer-vision based security platform that seamlessly integrates advanced object detection with dual-mode locomotion capabilities. Designed for diverse operational environments, the system supports both wheeled and quadrupedal configurations, ensuring effective performance on smooth indoor surfaces as well as challenging, uneven outdoor terrain. The architecture is organized into two primary subsystems: an intelligent computer vision module and a versatile locomotion unit.

The computer vision component employs a high-resolution camera coupled with deep learning algorithms, including YOLOv3, to detect and classify up to twelve distinct object types with an accuracy of approximately 70%. This capability enables the system to accurately identify potential intruders and communicate critical information to a centralized monitoring station, thereby enhancing situational awareness and prompt response. Simultaneously, the locomotion subsystem is engineered to deliver adaptive movement; the wheeled system provides rapid and efficient transit on flat surfaces, while the quadrupedal system offers superior maneuverability for navigating obstacles and irregular terrains.

Central to the design is a modular carrier board that not only facilitates seamless switching between locomotion modes but also consolidates the power and signal interfaces required by both subsystems. This integrated approach enhances reliability, simplifies maintenance, and allows for future scalability, including the potential integration of additional sensors and advanced control algorithms. Overall, ServoSentry stands as a robust, high-performance security solution, delivering comprehensive area surveillance and autonomous patrol capabilities, while remaining adaptable to the evolving needs of modern robotics applications.

Team Responsibilities

	Bryce Cadieux	Fola Adegoke	Noah Stieler	Peter Eshikena	Kazi Al Nahian	Seth Moule
Project Management			**P**		**P**	
Wheeled System				**S**	**S**	**P**
Quadrupedal System			**P**			
Carrier Board			**P**			**P**
Computer Vision					**P**	
Software Acceleration and Integration	**S**	**S**		**S**	**P**	
Wireless Communication	**P**				**S**	
Sensors (Cameras & Ultrasonic)		**P**		**S**	**S**	
Electrical System and Integration		**S**	**P**	**P**	**S**	**S**
Interface Board			**P**	**S**	**S**	
Decision Making Algorithm & Main Program	**S**	**S**		**P**	**P**	**S**
Legend: P – Primary responsibility, **S** – Secondary responsibility						

Acknowledgements

Group-08 would like to express their gratitude towards:

- Dr. Colin Gilmore and Dr. Ian Jeffrey for their motivation, patience, guidance, flexibility and financial support.
- Oleg Shevchenko for providing technical input and quick order placements.
- UMIEEE for their tools, and lounge/lab space to conduct initial meetings.
- For the mechanical aspect of the project, thanks to Zoran Trajkoski, for the help troubleshooting the entire mechanical design, providing the filament and handling all our 3D printing.
- Additionally, we would like to thank Zoran Trajkoski, Cory Smit, Gordon Glatz, Dr. Gilmore and Dr. Jeffrey for the invaluable discussions on improving the quadrupedal robot system.

List of Figures

Figure 2-1: High Level System Diagram 5

Figure 2-2 The three modules comprising the ServoSentry project. 6

Figure 2-3 High level block diagram of the electronics on the carrier board. Power connections from DC-DC converters are not shown. 7

Figure 2-4: Jetson NANO B01 Board Schematic [5] 8

Figure 2-5: Modified Jetson with the fan and Wi-Fi module installed with L-shaped USB and HDMI connectors for space constraints 9

Figure 2-6: FRIWOL USB HD1080P 2MP Webcam (Brixwm07-webcam) mounted on ServoSentry, with the broadcast indicator green, confirming active video feed 10

Figure 2-7: Raspberry Pi 3B+ board. [11] 11

Figure 2-8: HCSR04 Sensor [13] 12

Figure 3-1 The Flat Chassis Design with without the Carrier Board Magnet Holes 16

Figure 3-2 First Prototype of the Flat Chassis Fully Assembled Without Wiring 17

Figure 3-3 Final Design for the Flat Wheeled Chassis 17

Figure 3-4: Final Printed Version of the Flat Chassis 18

Figure 3-5: Wiring for the wheel system. 19

Figure 3-6 The first attempt at a leg design. Left to right: Side profile of CAD model. Isometric view of CAD model. Attempt at assembling this design, which failed because of the naive approach to making rotational joints. 21

Figure 3-7 Comparison between the initial failed joint, and the revised joint 22

Figure 3-8 Test of the leg supporting weight. 22

Figure 3-9 CAD models designed for the first walking test. 23

Figure 3-10 First test of the quadrupedal system standing. 24

Figure 3-11 CAD model of the final design of the leg. 25

Figure 3-12 Inverse kinematics. 26

Figure 3-13 Completed quadrupedal system. Left: Isometric view of the robot. The two loose cables are the power and signal connections to the carrier board. Right: Underside view of the robot. 28

Figure 3-14 Locations of the magnets in each module for securing the carrier board to the locomotion system. Note that for the quadrupedal system the magnets are inserted from the other side of the chassis. 29

Figure 3-15 Final CAD model of the carrier board with the subassembly sitting on top. 30

Figure 3-16 Front-facing and isometric views of the assembled carrier board. 30

Figure 4-1:Block Diagram Representing the Vision System Modules. 31

Figure 4-2: YOLOv3 Architecture Diagram showing how YOLOv3 splits the image into an S×S grid, with each grid cell predicting multiple bounding boxes and class probabilities [35] 33

Figure 4-3: YOLO detection network [4]. 33

Figure 4-4: Speed-accuracy trade-off for YOLOv3 [35]. 34

Figure 4-5: YOLOv3 detecting objects while showing detection accuracy and frame rate on our Jetson Nano 35

Figure 4-6: Vision System Detecting People using both the masked model and YOLOv3 COCO dataset. 37

Figure 4-7: Only the intruder detection model being implemented when a face covered individual is in the frame... 37

Figure 4-8: Screenshot showing an intruder and a non-intruder detected by the system. 39

Figure 4-9: Screenshot of the Raspberry Pi terminal receiving the bounding box intruder data sent from the Jetson, using UART. 40

Figure 4-10: Face recognition model successfully identifying an authorized user versus an unknown individual. 43

Figure 5-1: Block diagram showing UART wiring. 45

Figure 5-2: VPN Network 45

Figure 6-1: Ultrasonic sensors circuit diagram 47

Figure 6-2: Testing the ultrasonic sensor 25 cm away from a measuring tape. 48

Figure 6-3: Both ultrasonic sensors working and outputting obstacle distance successfully 49

Figure 6-4: Flowchart showing Pi processing of Vision and Ultrasonic data. 50

Figure 7-1 Power Distribution 51

Figure 7-2 Interface board and its pinout to the Raspberry Pi and locomotion systems. 53

Figure 8-1: High-Level System Diagram of ServoSentry, illustrating the data flow between the Jetson Nano, Raspberry Pi, ultrasonic sensors, and locomotion subsystems. 56

Figure 9-1: Final Integrated Wheel System 60

Figure 9-2: Final Integrated Quadrupedal System 60

Figure 0-1: YOLOv3 Loss and mAP curves from Darknet's training process. The decreasing loss and increasing mAP indicate successful training and improved object detection performance [37] 66

Figure 0-2: Loss and mAP curves for the mask detection model, indicating successful convergence over 6000 iterations [36] 67

Figure 0-3: The Pipeline of the MTCNN Framework that Includes Three-stage Multi-task DNNs [40]. 67

List of Tables

Table 1-1: Final Budget. 3

Table 3-1: Description of the interface provided by the LegControl class. 28

Table 4-1: Key Performance Parameters of Computer Vision Models 38

Table 7-1: Voltage and current specifications for key components....52

Table 9-1: Obstacle Avoidance Test Results 58

Table 9-2: Table of Performance Metrics 62

List of Abbreviations

CAD – Computer-Aided Design

UART – Universal Asynchronous Receiver/Transmitter

SBC – Single Board Computer

AI – Artificial Intelligence

API – Application Programming Interface

GPU – Graphics Processing Unit

HD – High Definition

USB – Universal Serial Bus

YOLO - You Only Look Once (a popular object detection system)

IoU – Intersection over Union

FPS – Frames per Second

MTCNN – Multi-task Cascaded Convolutional Networks

CNN – Convolutional neural network

SVC – Support Vector Classifier

NMS – Non-Maximum Suppression

ROS – Robot Operating System

COCO – Common Objects in Context

MAP – Mean Average Precision

PWM – Pulse Width Modulation

Table of Contents

Abstract i

Team Responsibilities ii

Acknowledgements iii

List of Figures iv

List of Tables vi

List of Abbreviations vii

1 Introduction 1

1.1 Project Overview 1

1.2 Motivation 2

1.3 Scope 2

1.4 Budget 2

1.5 Report Organization 4

2 System Architecture, Hardware and Sensors 5

2.1 Hardware Components 5

2.2 Vision Processing Board 7

2.3 Camera Used for Object Detection 10

2.4 Main Control Hub 11

2.5 Ultrasonic Sensor 12

2.6 Software Stack 13

3 Mechanical Design 15

3.1 Wheeled System 15

3.1.1 Motors, Wheels and Drivers 15

3.1.2 The Flat Chassis Design 16

3.1.3 Wiring 18

3.1.4 Wheel Control 19
3.2 Quadrupedal system 20
3.2.1 Leg Geometry and Motor Selection 20
3.2.2 Design of Rotational Joints 21
3.2.3 Design of Chassis and Rotation of the Entire Leg 22
3.2.4 A First Attempt at Walking 23
3.2.5 Design of Leg Structural Components 24
3.2.6 Inverse Kinematics 25
3.2.7 Walking Algorithm 27
3.2.8 Final Outcome 27
3.2.9 Integration and Programming Interface 28
3.3 Carrier Board 29
4 Computer Vision System 31
4.1 Why Masked-Person Detection? 31
4.2 Object Detection 32
4.2.1 YOLOv3: Real-Time Object Detection 32
4.2.2 Object Detection Pipeline in ServoSentry 34
4.3 Intruder Detection 35
4.3.1 Detection Pipeline 36
4.4 Bounding Box Data Processing 38
4.4.1 Remote Access 40
4.5 Facial Recognition 40
4.5.1 Face Recognition Model 41
4.5.2 Model Selection and Training Process 41
4.5.3 Challenges with Integration 43
5 Communication System 44

5.1 Jetson Nano to Raspberry Pi Communication 44
5.1.1 UART Configuration 45
5.2 Wi-Fi Module and Networking 45
6 Ultrasonic Sensors 47
6.1 Concurrent Data Processing with Computer Vision 49
7 Electrical System 51
7.1 Interface board 52
8 Software Implementation 54
8.1 Main control code description 54
8.2 Decision-Making Algorithm 55
9 Integrated System Test and Results 58
9.1 Obstacle Avoidance Tests 58
9.2 Intruder Detection and Tracking 59
9.2.1 Wheel System 59
9.2.2 Quadrupedal System 60
9.3 Performance Metrics 61
10 Future Enhancements 63
10.1 Wheeled System 63
10.2 Quadrupedal System 63
10.3 Vision System 64
11 Conclusion 65
Appendices 66
Appendix-A 66
Appendix B. Code Listings and Test Data 68
Reference List 69

1 Introduction

1.1 Project Overview

ServoSentry is a modular robotics platform with computer-vision based target identification for planning the movement of the robot. It is designed to be a versatile security robot that can navigate multiple environments while detecting threats and potential intruders. Although we call it “ServoSentry,” the term “Sentry” is primarily a nod to one possible application: an autonomous security patrol. However, our primary motivation was to create a fun, versatile robot platform that can address a variety of tasks beyond surveillance. We explored multiple use cases during development. For instance, with appropriate modifications the robot can:

- Patrol an indoor area to keep track of unusual activity,
- Assist in search-and-rescue exercises in constrained environments,
- Entertain or guide visitors in a museum or campus tour scenario,
- Carry out basic object detection and tracking for educational demonstrations.
- Serve as a children’s toy companion.

To make our robot adaptable across different terrains and requirements, we incorporated two distinct locomotion systems: a wheeled platform for speed and maneuverability on flat surfaces, and a quadrupedal system for more complex terrain like navigating over obstacles or uneven surfaces. Both systems share a common carrier board that houses all the electronics, making it quick and easy to swap between locomotion modes.

Beyond the locomotion versatility, our platform integrates a robust computer vision pipeline running on an NVIDIA Jetson Nano, enabling object detection and optional face recognition tasks. We use a Raspberry Pi as the main control hub for sensor data and actuation. This modular approach allows us to demonstrate a wide range of applications, from basic obstacle avoidance to more advanced tasks like following an intruder or investigating unusual activity.

1.2 Motivation

Modern robotics can serve a wide range of applications, from warehouse inventory tracking and environmental exploration to educational demonstrations, yet many existing solutions are constrained by single-locomotion designs or limited perception. Our project aims to create a flexible, multi-locomotion robot equipped with advanced computer vision for real-time detection and navigation. While we illustrate its capabilities through a "sentry" use case (for example, detecting intruders), the underlying architecture is broadly applicable: it can be adapted for object tracking, inspection, and autonomous patrolling across diverse environments. This versatility, coupled with a modular design for swapping locomotion systems, makes our robot a powerful platform for both practical deployments and research-oriented experimentation.

1.3 Scope

The ServoSentry system is designed with several integrated systems, each with distinct performance measures to achieve functionality. The computer vision module focuses on object detection and recognition, with speed and accuracy indicators. In software, object classification and facilitation of real-time data transmission between the computer vision and control system is handled. The sensor module involves measuring certain parameters to evaluate the range of detection. For the communication module, wireless connectivity to enable remote access is implemented with indicators assessing the stability of the transmissions. Power supply is evaluated on the runtime across the different locomotion states. Both quadrupedal and wheeled systems, supporting smooth and adaptable movement over a certain load capacity. Finally, integration ensures system adaptability with the ease of transitioning between the locomotion states.

1.4 Budget

The budget of ServoSentry amounted to a total $1200 with $600 being supplied by the Price Faculty of Engineering, and $300 being supplied by both Colin Gilmore and Ian Jeffrey. The “*” represents items that were kindly donated for the project either by a team member or the ECE Tech Shop. The planned budget was estimated to be $888.95 with the final budget ending up being $1154.36.

Table 1-1: Final Budget.

Item	Source	Quantity	Unit Cost	Total Cost (including tax and shipping)
Nvidia Jetson Nano Developer Kit	Amazon	1	$239.00	$307.00
Gear Motor and Wheel Kit	Amazon	3	$33.26	$111.75
DC to AC Converter Power Supply	Amazon	1	$15.99	$17.91
Motor Drivers	Amazon	1	$16.99	$19.03
PCA9685 servo driver	Amazon	1	$15.99	$18.07
Ball bearings 30pcs 5mm x 14 mm x 5mm	Amazon	1	$13.99	$15.81
Assorted metric screw kit	Amazon	1	$26.99	$30.50
Magnets, 6x3mm 100 pieces	Amazon	1	$13.99	$15.81
DC-DC Converter 6-40V input, 1.2-36V output	Amazon	2	$16.99	$38.40
LiPo battery, 2200mAh, 35C	Amazon	1	$53.99	$61.02
XT60 battery connector	Amazon	1	$9.99	$11.29
Battery charger	Amazon	1	$14.99	$16.94
PLA 3D printer filament, purple	Amazon	3	$27.99	$94.89
Assorted metric screw kit	Amazon	1	$26.99	$30.50
Dedicated Cooling Fan for Jetson Nano	Amazon	1	$12.93	$13.84
3 POS WAGO Connectors (10 Pack)	Amazon	1	$11.75	$13.16
DS3225 Hip and knee rotation motors	Amazon	3	$72.99	$247.44
Micro usb male pigtail connector	Amazon	1	$12.99	$14.68
PCA9685 servo driver	Amazon	1	$15.99	$18.07
Twippo Connector Terminals	Amazon	1	$15.99	$17.11
PCA9685 servo driver	Amazon	1	$15.99	$18.07
L Shaped USB Connector	Amazon	1	$13.99	$14.97
L Shaped HDMI Connector	Amazon	1	$7.59	$8.12
MG996R Leg rotation motors	Team Member	*	$0.00	$0.00
Raspberry Pi 3B+	ECE Tech Shop	*	$0.00	$0.00
			Overall Cost	$1,154.36
			ECE provided funds	$600
			Donation from Advisors	$600
			Net remaining funds	$45.64

1.5 Report Organization

The remaining sections of the report are organized as follows:

- Chapter 2: high-level overview of the components that make up ServoSentry.
- Chapter 3: mechanical design of ServoSentry.
- Chapter 4: ServoSentry's computer vision system.
- Chapter 5: communication system for ServoSentry.
- Chapter 6: ServoSentry's obstacle detection using ultrasonic sensors.
- Chapter 7: electrical and power system of ServoSentry.
- Chapter 8: ServoSentry's software implementation.
- Chapter 9: testing of ServoSentry.
- Chapter 10: future enhancements to ServoSentry.
- Chapter 11: conclusion.

2 System Architecture, Hardware and Sensors

ServoSentry's flow of input data to an actuation of the locomotion system is illustrated in Figure 2-1. The data from the computer vision system and ultrasonic sensors is used by the Raspberry Pi to determine the movement of the robot. If computer vision detects a person, then the Raspberry Pi would take that information, and control the locomotion system to follow that person. The Raspberry Pi is mounted on the carrier board (along with all other electronics) and this module is what we swap between locomotion systems.

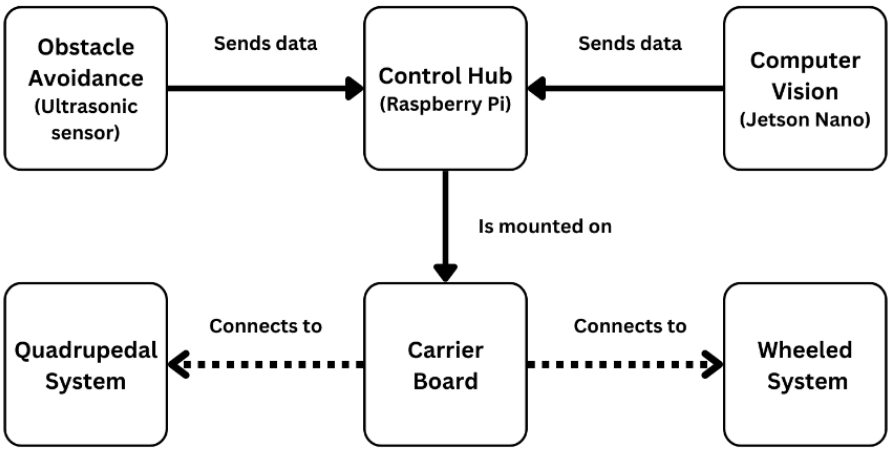


Figure 2-1: High Level System Diagram

2.1 Hardware Components

The hardware architecture is centred around three modules:

- The wheeled locomotion system.
- The quadrupedal locomotion system.
- The carrier board.

All of which are showcased in Figure 2-2. The carrier board is a 3D-printed assembly that all our electronics, sensors, and battery mount onto. To switch the carrier board between wheeled or quadrupedal systems, the user simply must place the carrier board on the locomotion system,

connect the power and signal wires for the motors, and flip the toggle switch to the desired system.

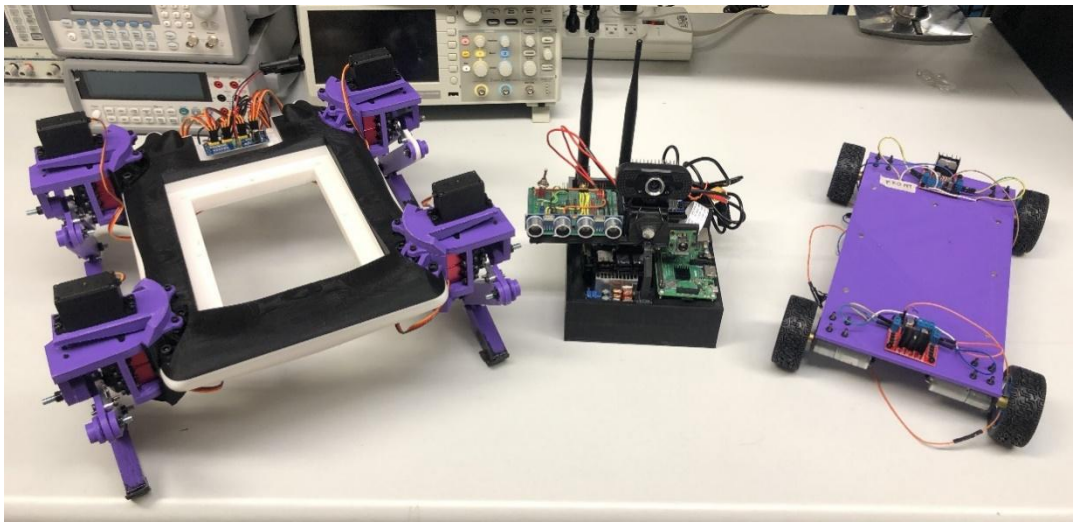


Figure 2-2 The three modules comprising the ServoSentry project. From left to right: quadrupedal system, carrier board, and wheeled system.

Each locomotion system is designed to move forwards, turn, and securely transport the carrier board. The wheeled locomotion system consists of four DC motors, and two DC motor drivers, which all mount onto a 3D printed chassis. Similarly, the quadrupedal system has four 3D printed legs, and each leg contains three servo motors making twelve motors overall. This system also has a 3D printed chassis, and a servo motor driver mounted on it. A motor driver is a specialized electrical circuit, that is used to control electric motors. They typically have a high-power input, and a collection of signal inputs that a microcontroller would interface with to control the motors.

The core of the electrical system is a Raspberry Pi, which is responsible for receiving data from all our sensors, then using that data to control a locomotion system, for example by avoiding an incoming obstacle, or following an identified target. Aside from the motor drivers, all our electrical hardware is mounted onto the carrier board. Figure 2-3 illustrates the block diagram for the system. The Raspberry Pi interfaces with all the peripheral devices, most importantly of which is the computer vision system, consisting of a Nvidia Jetson Nano and computer webcam. To bring all our systems together seamlessly, we have an interface board which handles two ultrasonic sensors, a toggle switch for locomotion system selection, and pin interfaces for quickly connecting locomotion systems. The ultrasonic sensors provide the robot with the distance between it and potential obstacles in front of it, which is used to implement

basic obstacle avoidance. To power everything, we are using a 11.1V LiPo battery and two buck converters, which converts the 11.1V to either 5V for the Raspberry Pi and Nvidia Jetson, or 6V for the locomotion systems.

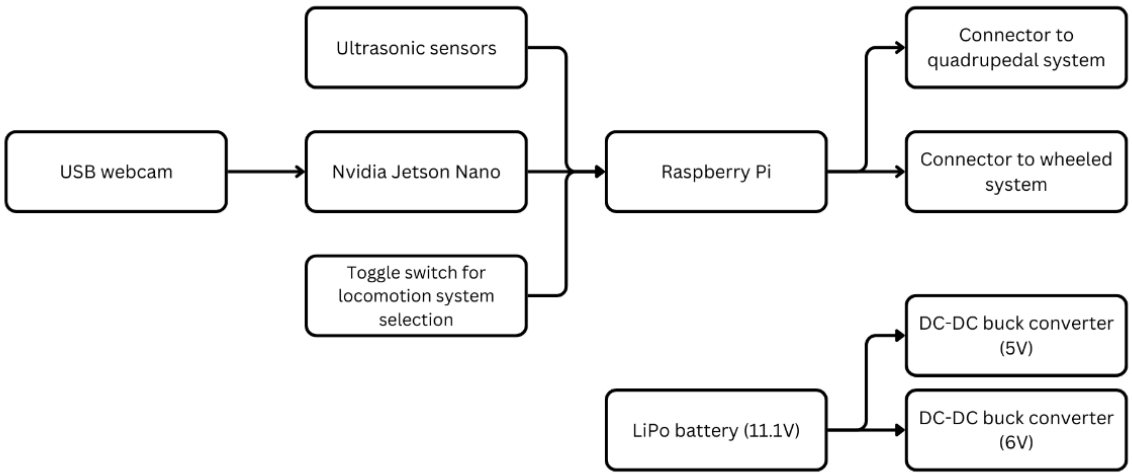


Figure 2-3 High level block diagram of the electronics on the carrier board. Power connections from DC-DC converters are not shown.

2.2 Vision Processing Board

The vision processing component of ServoSentry is built around the OKdo Nano Developer Kit, powered by the NVIDIA® Jetson Nano module [1]. Designed specifically for edge AI applications [2], the Jetson Nano is responsible for running deep learning models and processing real-time video feeds captured by the connected USB camera. Its efficient CUDA-enabled GPU [3] enables rapid execution of YOLOv3-based object detection [4], which is critical for the robot's autonomous security functions.

Key specifications of the board include a 128-core Maxwell GPU, a quad-core ARM Cortex A57 CPU running at 1.43 GHz, and 4 GB of 64-bit LPDDR4 memory. These components ensure that high-resolution video streams are processed effectively, while the board's extensive I/O features multiple USB ports, Gigabit Ethernet, and dual MIPI CSI-2 camera connectors. However, in our implementation, the camera is connected via USB. Figure 2-4 shows the schematic of the Jetson Nano board used in this project.

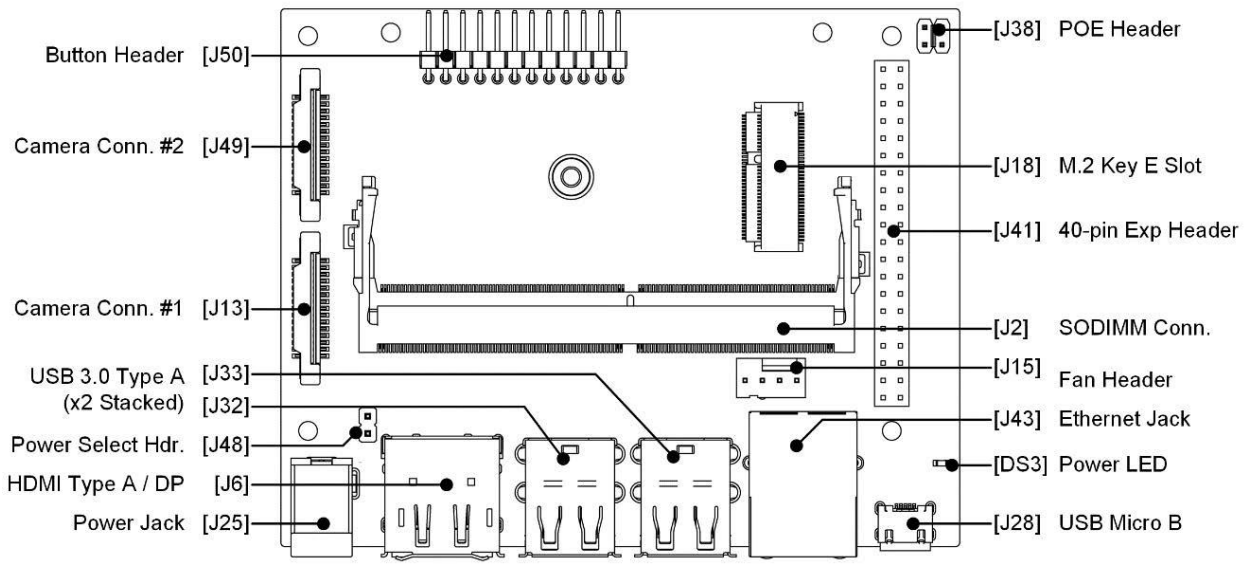


Figure 2-4: Jetson NANO B01 Board Schematic [5]

To maximize performance, the board is configured with NVIDIA's JetPack SDK [6], which provides essential libraries such as CUDA, cuDNN [7], and TensorRT [8]. Essential dependencies like OpenCV [9] have been installed, and the object detection script is set to load automatically at startup, ensuring that the system operates autonomously upon power-up. Moreover, to prevent thermal throttling during extended operation, a cooling fan was installed and configured using the Pyrestone jetson-fan-ctl tool. This dynamic cooling solution adjusts fan speed based on temperature readings, ensuring stable performance under load. Additionally, a separate WiFi module has been installed on the device to allow remote access to the vision system when the board is mounted on the locomotion system. Figure 2-5 shows the Jetson Nano board and the WiFi antennas after modification.

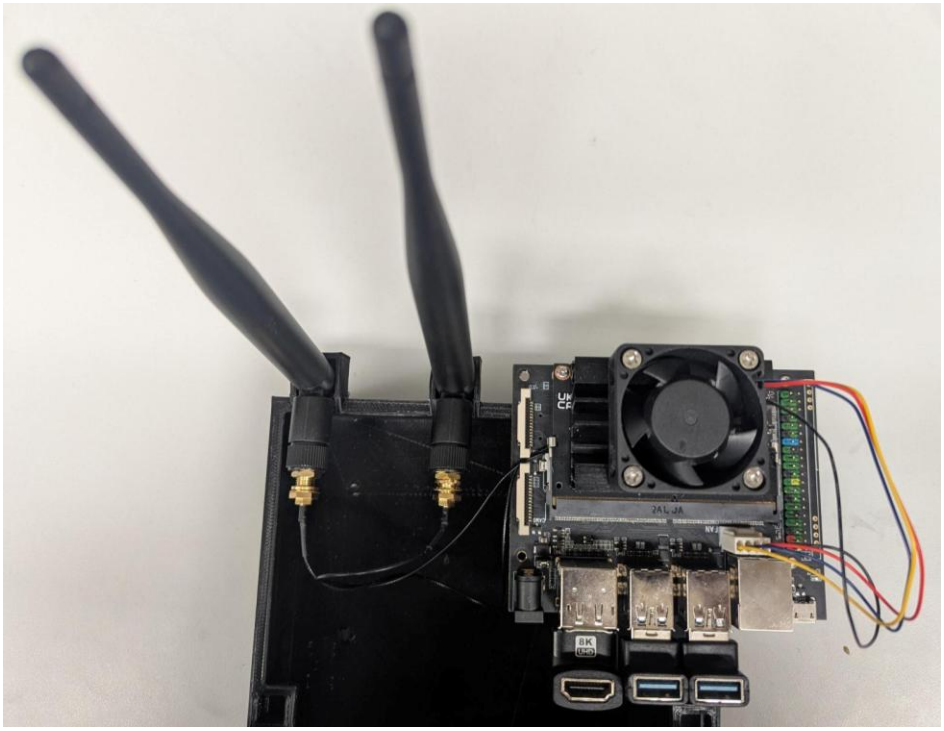


Figure 2-5: Modified Jetson with the fan and Wi-Fi module installed with L-shaped USB and HDMI connectors for space constraints

List of Definitions and References of Terms Used Above:

1. **Jetson Nano:** A compact, powerful computer module designed for deploying AI applications at the edge. It features a GPU optimized for parallel processing and deep learning tasks. [1]
2. **Edge AI:** Running artificial intelligence algorithms directly on a device ("the edge") instead of in a centralized cloud, enabling faster, real-time decision-making. [2]
3. **CUDA (Compute Unified Device Architecture):** A parallel computing platform and programming model from NVIDIA that enables developers to use GPUs for general-purpose processing, significantly accelerating computation. [3]
4. **YOLOv3 (You Only Look Once, Version 3):** A real-time object detection system that processes an entire image in one forward pass through a deep neural network to quickly and accurately detect objects. [4]
5. **JetPack SDK:** NVIDIA's integrated software development kit for the Jetson family of embedded computing modules. [6]

6. **cuDNN (CUDA Deep Neural Network library):** A GPU-accelerated library that provides optimized implementations for deep neural network routines such as convolutions, pooling, and activation functions. [7]
7. **TensorRT:** An inference optimizer and runtime library by NVIDIA designed to optimize deep learning models for deployment by reducing latency and increasing throughput during inference. [8]
8. **OpenCV (Open-Source Computer Vision Library):** A widely used open-source library that provides tools for computer vision, machine learning, and image processing. [9]

2.3 Camera Used for Object Detection

This camera was available to be used in our project at no cost. It employs a CMOS sensor to capture high definition 1080p video, ensuring that the input to our YOLOv3 object detection pipeline is of sufficient quality for accurate analysis. Its design includes a 30-degree up-and-down rotation capability, which allows for flexible positioning to optimize the field of view according to the mounting configuration [10].

A feature of this camera is its integrated broadcast indicator. As soon as the Jetson Nano responsible for running the object detection algorithms powers up the camera quickly initializes and begins streaming video. The green broadcast symbol on the camera serves as an immediate visual cue that the detection feed is live, confirming that the object detection algorithm is actively processing frames. This automatic activation is essential for ensuring the system operates autonomously without manual intervention, thus enhancing overall reliability.

Figure 2-6: FRIWOL USB HD1080P 2MP Webcam (Brixwm07-webcam) mounted on ServoSentry, with the broadcast indicator green, confirming active video feed

2.4 Main Control Hub

The control hub is the central system that connects and coordinates the vision and locomotion systems. It processes the data from sensors, makes decisions based on that data, and directs movements and actions, enabling ServoSentry to function autonomously as a guard robot. As such, the control hub would need to be device capable of handling the computing processes ServoSentry requires.

An initial choice for the control hub was the Jetson Nano board. This meant the Jetson Nano would be responsible for processing sensor data as well as the actuation of the respective locomotion systems. However, this would require a lot of power which the Jetson Nano would not be able to provide for both tasks. As a result, another single board computer (SBC) the Raspberry Pi 3B+ was chosen to serve as the central system for actuation and sensor data processing as shown in Figure 2-7: Raspberry Pi 3B+ board. [11].

Figure 2-7: Raspberry Pi 3B+ board. [11]

The key specifications of the Raspberry Pi are highlighted as follows [12]:

- 64-bit quad core processor which runs at 1.4GHz.
- 1GB memory.
- 4 USB 2.0 ports.
- 5V/2.5A micro-USB connector for power.
- 2.4GHz and 5GHz wireless Local Area Network.
- CSI camera port.

2.5 Ultrasonic Sensor

Ultrasonic sensors were selected for our project primarily due to their cost-effectiveness and ease of integration. Additionally, they were available to be borrowed from the University's ECE department tech shop which saved us the funds to buy them. These sensors offer reliable short-range distance measurements; essential for obstacle detection in dynamic environments while maintaining a low cost compared to more complex systems such as LiDAR.

The HC-SR04 ultrasonic sensor we are using is a widely used device for non-contact distance measurement, offering a range from 2 cm to 400 cm with an accuracy of approximately 3 mm. It operates by emitting ultrasonic pulses at 40 kHz and measuring the time it takes for the echo to return after reflecting off an object. This time-of-flight measurement is then used to calculate the distance to the object. It has the following feature:

- **Operating Voltage:** 3.3V DC
- **Working Current:** 15 mA
- **Measuring Angle:** 15 degrees
- **Trigger Input Signal:** 10 µs TTL pulse
- **Dimensions:** 45.5 x 20 x 15.5 mm [13] [13]

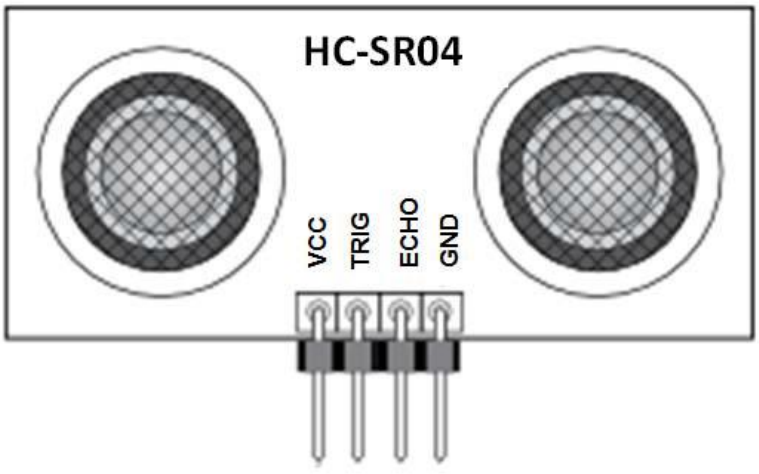


Figure 2-8: HCSR04 Sensor [13]

The decision to integrate the HC-SR04 sensor with the Raspberry Pi was influenced by two factors:

1. **Cost-Effectiveness:** The HC-SR04 provides reliable distance measurements at a low cost, making it suitable for projects with budget constraints.
2. **Ease of Implementation:** Its straightforward interfacing requirements and widespread documentation facilitate rapid development and integration.

2.6 Software Stack

The project is based around the Nvidia latest JetPack SDK supported by the aarch64 Jetson Nano, version 4.6.4. JetPack 4.6.4 uses Jetson Linux 32.7.6 which includes [6] [14]:

- **the bootloader:** for initializing the hardware,
- **Linux kernel version 4.9.337**, [15]: the core operating system for interfacing between hardware and the processes,
- **Ubuntu desktop environment 18.04**, [16]: the graphical user interface,
- **CUDA 10.2**, [17]: for interfacing with the computer vision graphics acceleration hardware used,
- **drivers**: for interfacing with external peripherals, such as the USB camera, and
- **the toolchain**: for compiling software compatible with the device.

Beyond these dependencies, the project's software primarily runs on Python 3.6 which uses the following dependencies [18]:

- `numpy 1.19, [19]`: for efficient math calculations used in the computer vision algorithms,
- `opencv-python 4.10.0.84, [20]`: for manipulating the webcam livefeed,
- `matplotlib 3.9.4, [21]`: for annotating images,
- `pyserial 3.5.0, [22]`: for serial communication with the Raspberry Pi,
- `pytorch 1.12.1, [23] ; torchvision 0.13.1, [24] ; scikit-learn 1.6.1, [25]:` for training images for facial recognition.
- `pillow 11.1.0, [26]`: for image manipulation,
- `joblib 1.4.2, [27]:` for exporting and importing python objects,

Additionally, OpenCV 4.8.0 was compiled from source with dedicated hardware support and python bindings [28]. This allows for accelerated processing to meet the project requirements. Outside of the libraries used, YOLOv3 was the primary AI model used for image recognition and facenet-pytorch was the primary model used for face recognition [29] [30].

3 Mechanical Design

As a starting point for our locomotion designs, we estimated how much weight the system must be able to move based on approximately chosen dimensions for a 3D printed chassis, and the weight of our chosen electrical hardware. The estimate we came to was 1.75 kg, and we designed the wheeled and quadrupedal systems with the intent of supporting at least 1.75kg.

3.1 Wheeled System

The wheeled aspect of the robot was designed to be the faster and more manoeuvrable of the two locomotion systems but only capable use on a single floor of a building. It was designed to be used for situations such as contained construction yards, one floor museum wings and similar applications. The design uses simple dc motors, one for each wheel, powered and controlled by motor drivers. Each wheel was connected to the motor using a coupling with the wheel screwed onto the coupling and the motor and wheel being mounted on a mounting bracket to some form of chassis. As such, a total of four wheels were used along with motors and two motor drivers.

3.1.1 Motors, Wheels and Drivers

The motors, wheels and drivers were selected to be simple to assemble. For the motors, we decided on using the 25GA370 280rpm 6V Brush DC Gear Motors given that they came with mounting brackets, couplings as well as 65mm wheels which suited our needs as the size of the wheels matched with the size of our robot being about the size of a laptop. Furthermore, the gear motors granted 0.9Nm of torque per motor which would be able to carry our maximum estimated load of 1.75kg which would end up being the approximated load of the electronics.

As for the motor drivers, we used the BOJACK L298N Motor Driver Controllers. These controllers allowed us to use two DC gear motors per driver and could vary the speed of the motors for a more advanced design or it could just make the motors go forward and backward if more advanced control was not required. Furthermore, the drivers were able to use the same voltage source for both the controller and the motors given that the motors did not use more than 12V which would have required the use of two different voltage sources.

3.1.2 The Flat Chassis Design

The goal behind the flat chassis design was to keep it simple and as such quick to modify and test if needed. The flat design ended up being 160x260 mm and had the holes for the mounting brackets on all four corners of the chassis with the holes for motor drivers on the left and right of the chassis, with a couple of small holes to fish the wires from the motors to the motor drivers through as seen in Figure 3-1.

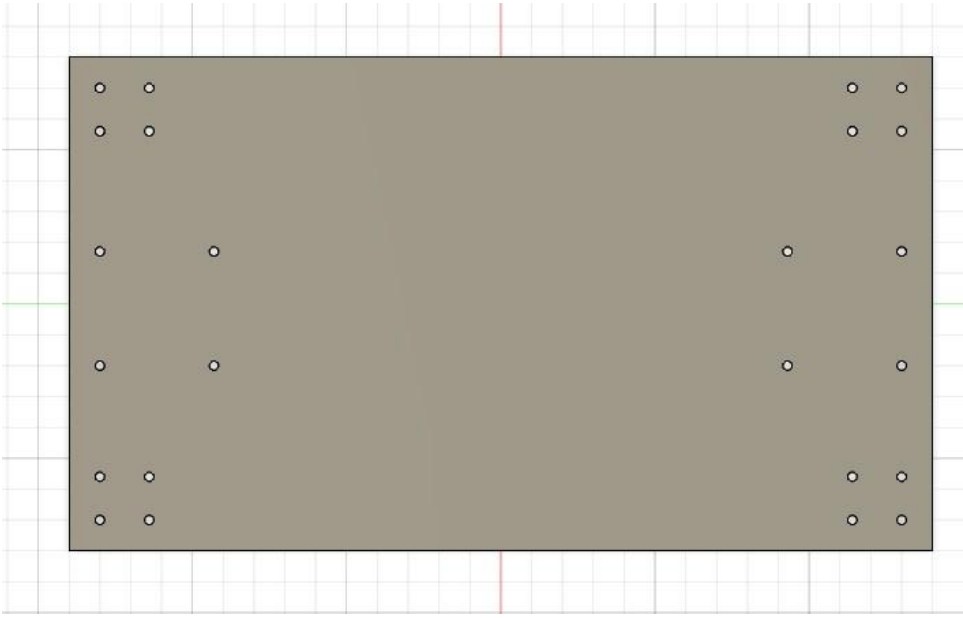


Figure 3-1 The Flat Chassis Design withoutt the Carrier Board Magnet Holes

This was the first chassis that was sent to manufacturing. Initially it was going to be 3D printed however, given its simplicity, we were advised to just get an acrylic board cut instead. When fully assembled the chassis, seen in Figure 3-2, with some initial tests applying 8V was able to move at a speed of at least 0.25m/s which met our speed requirements while carrying a load of approximately 1.5kg which met our weight requirements. The chassis was then sent back to the drawing board to add in the mounts for the carrier board.

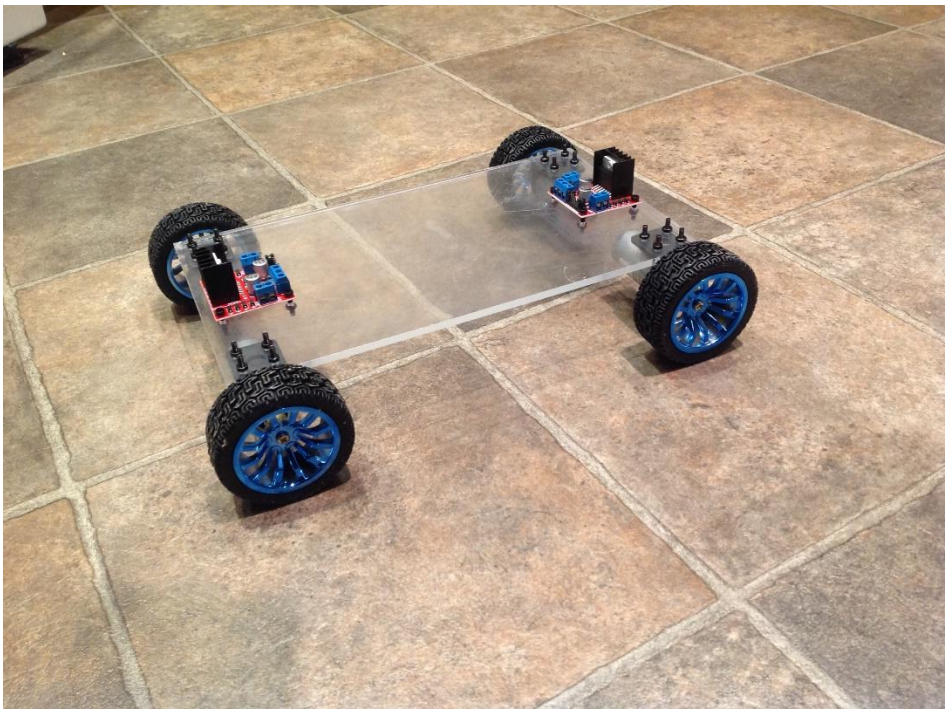


Figure 3-2 First Prototype of the Flat Chassis Fully Assembled Without Wiring

When it was decided that the mounting method was going to be magnets embedded at the bottom of the carrier board and at the top of the chassis. The chassis was redesigned to add in these holes at exactly spaced intervals around the edge of where the carrier board would sit, with a total of eight indents. Furthermore, the chassis was made 40mm larger to be able to more effectively hold the carrier board. Finally, the size of the holes to fish the wires through was increased to accommodate more wires as seen in Figure 3-3.

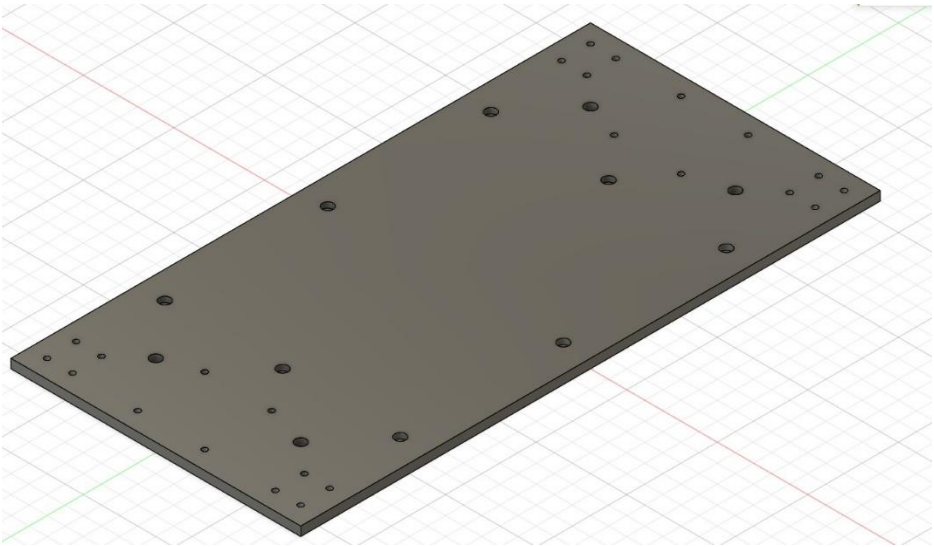


Figure 3-3 Final Design for the Flat Wheeled Chassis

The final iteration of the flat chassis was 3D printed and then assembled. One issue arose when the magnets were being embedded as one of them was put in the wrong way and had to be dug out. Afterwards, the hole that remained was too big to stick another magnet in, but the other seven magnets had no issue in keeping the carrier board properly attached. The final chassis was then completed other than minor adjustments to the wiring as can be seen Figure 3-4. This new design also met the design parameters, able to move at 0.52m/s and carry a weight of 1.75kg which meet both our weight and speed requirements.

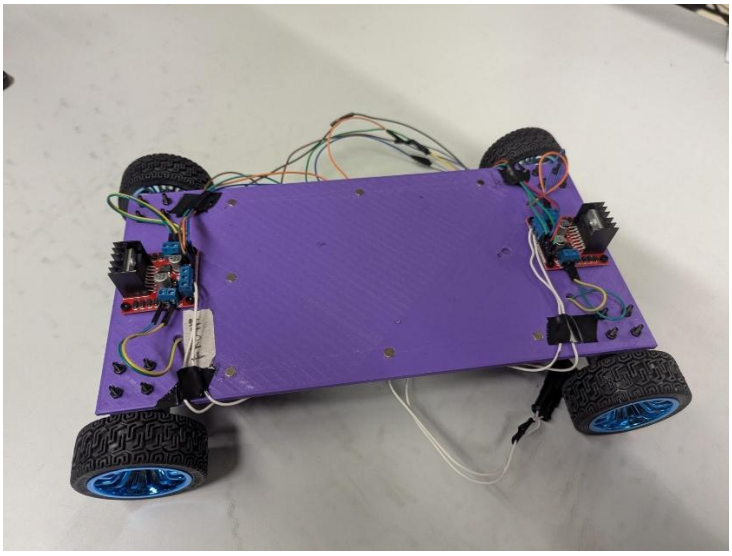


Figure 3-4: Final Printed Version of the Flat Chassis

3.1.3 Wiring

To ensure the functionality and safety of the robot, proper wiring is crucial. The wheel system is powered by an 11.1V LiPo battery and various GPIO pins that allow for motor control via motor drivers. A diagram of the wiring of the system is show Figure 3-5:

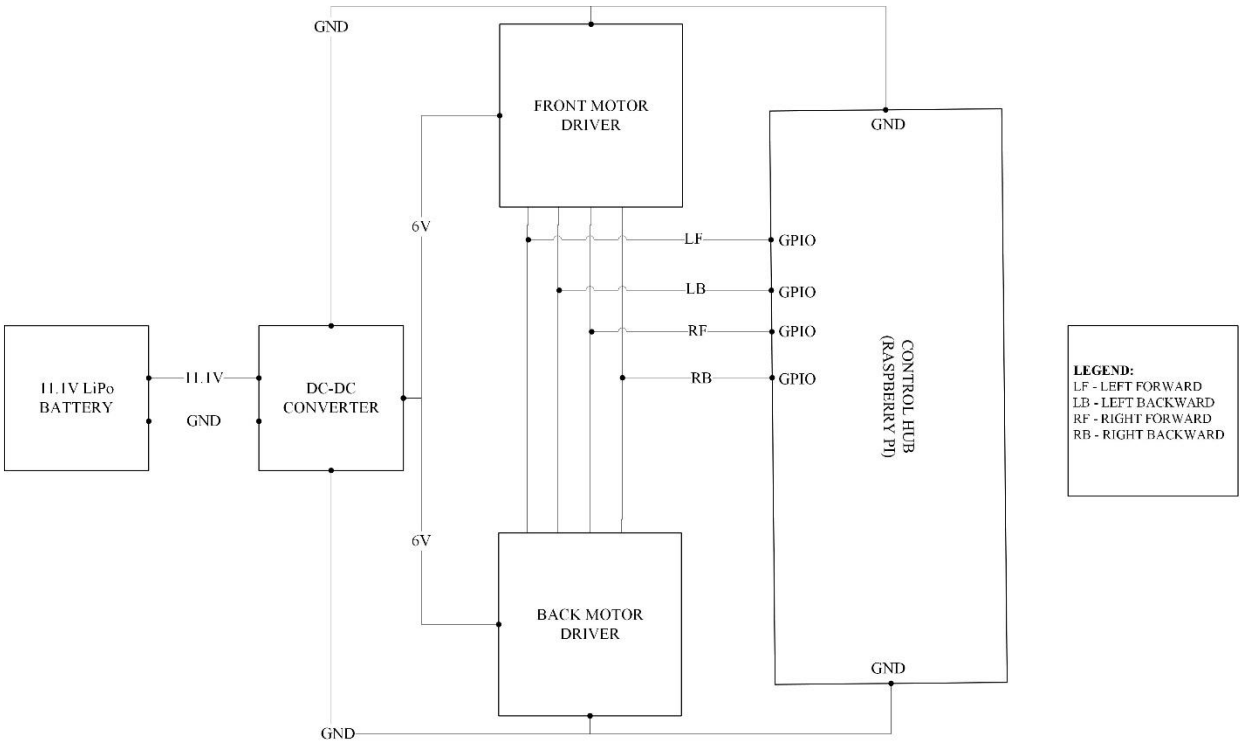


Figure 3-5: Wiring for the wheel system.

The DC-DC converter steps down the 11.1V from the LiPo battery to 6V to power the motor drivers. The front motor driver and the back motor driver have their inputs daisy chained to reduce the number of GPIO pins required by the Raspberry Pi. The LF, LB, RF and RB lines referenced in Figure 3-5 are the input pins of the motor driver that connect to the Raspberry Pi.

3.1.4 Wheel Control

ServoSentry's wheel control system uses an open-loop control system, meaning that there is no feedback mechanism to adjust the speed of the wheels. The wheels run at a constant speed when activated as the code directly sets the GPIO pins to high or low without making use of Pulse Width Modulation (PWM) signals for speed regulation. The system relies on dedicated functions for each movement task. Forward, backward, left and right turns are the different movements the wheel control is capable of.

While this open-loop design is simple and cost-effective for straightforward movement tasks, the wheel control could be enhanced in the future by integrating PWM for variable speed control resulting in a more stable system since movements would be less erratic.

3.2 Quadrupedal system

The design and fabrication of quadrupedal robots is a highly interdisciplinary area of technology that requires proficiency in many fields like control systems, electronics, and machine design. Thus, when attempting a project in robotics, the most important decision is to set the scope to something that aligns with the aspiring roboticists' skillset and resources. To narrow down this module's scope, we consulted other amateur quadrupedal robot projects to get a starting point for leg geometries, control systems, and construction techniques. James Bruton's openDog [31] project was a major inspiration and one of the best amateur quadrupedal robots as he implemented custom designed mechanical assemblies and balancing capabilities. The geometry for our leg design however was based on projects similar to those in [32] and [33], as these projects are much simpler than openDog, making them more realistic to use as a reference for our project, given our skillset and time constraints.

The objective of this module was simply to design and fabricate a quadrupedal robot that can support the load required 1.75 kg without the legs failing, and to walk forwards and turn. Every structural component in the system excluding the motors and mechanical hardware (nuts, bolts, bearings, etc.) were custom designed in Fusion 360 and 3D printed with PLA filament.

The resulting quadrupedal system met all our movement and mechanical goals, except for achieving a movement speed of 25 cm/s. Instead, our design attained a final average speed of 1.63 cm/s which did not meet our requirement. In addition, we created a Python package to control the quadrupedal system, which made integration into the rest of ServoSentry simple and easy. This section is structured to walk the reader through all aspects of the final design by using our previous design iterations to illustrate problems we faced, and how we solved them. Each iteration of the quadrupedal system highlights an aspect of the final system design.

3.2.1 Leg Geometry and Motor Selection

The first goal for the design of this system was to develop a leg that could support the load required by our project, without the leg failing. While achieving this first goal, we would also be able to address initial issues with 3D printing and leg joint assemblies before committing to fabricating all four legs. The entire leg design is based around our choice of motors, which we selected using our load requirement of 1.75 kg.

We decided on using 25 kg/cm DS3225 servo motors for the motors comprising the leg, as this specific motor was cheap while offering high torque. Servo motors were used because they have high torque and a simple built in feedback control system that enables the motor's output shaft to be set to a desired angle, which is required for robotics. To control the motors, a 16-channel PCA9685 based servo driver board was used. A 3D model of the first leg prototype was then designed as shown in Figure 3-6. This geometry is kept throughout the entire project and provides two degrees of freedom (DOF) meaning the robot's foot can move freely within the plane, a requirement for being able to lift the leg and take steps.

This design was fabricated but assembly failed due to the rather naïve approach taken to building the rotational joints, as the bearing was embedded in one part, while the other part had a 3D printed plastic peg that would get inserted into the bearing. Figure 3-6 shows that this joint was weak to the point that it snapped upon attempted assembly. This design failure was likely a consequence of using 3D printed PLA as parts are constructed in layers, which makes them weaker to forces applied tangential to these layers.

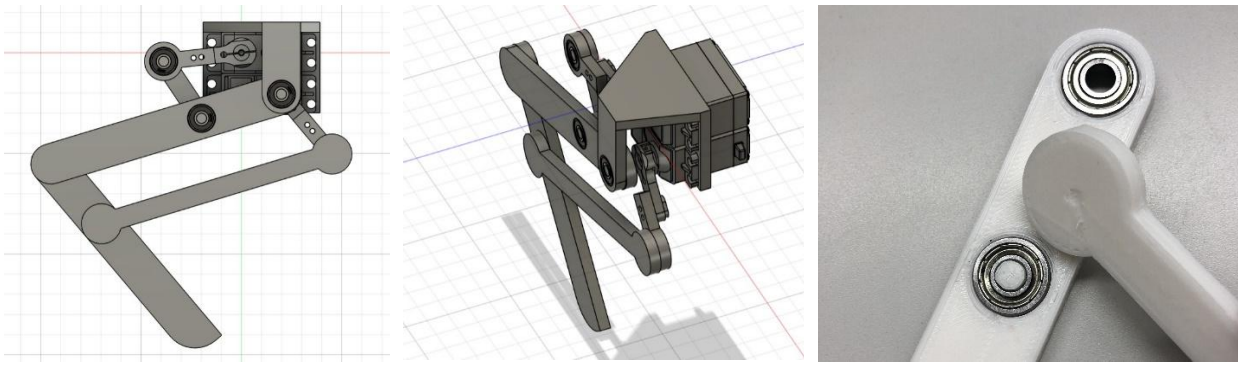


Figure 3-6 The first attempt at a leg design. Left to right - Side profile of CAD model. Isometric view of CAD model. Attempt at assembling this design, which failed because of the naive approach to making rotational joints.

3.2.2 Design of Rotational Joints

To address this issue, the final leg joints were designed with a metal hexagonal bolt that would go through both parts and be secured to the bearing with a nut. Figure 3-7 demonstrates the difference between the two joints. This new revision was successfully assembled, and a weight test was performed in Figure 3-8. The test indicated that the leg was able to support at least 750g of mass indicating that all four legs would be able to support 3kg, which is well above our target of 1.75kg. This test gave us confidence to fabricate the remaining three legs and chassis to carry on with a walking test.

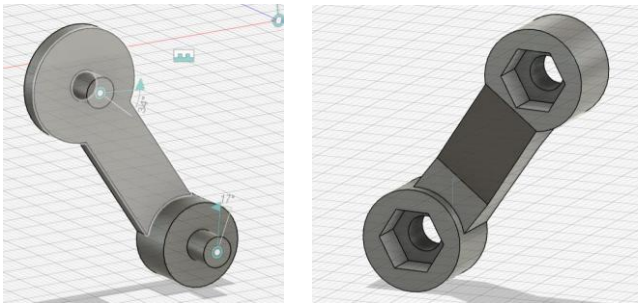


Figure 3-7 Comparison between the initial failed joint, and the revised joint. The hexagonal cutout in the part is where the hexagonal bolt slots into, preventing it from rotating with respect to the part it is embedded in.

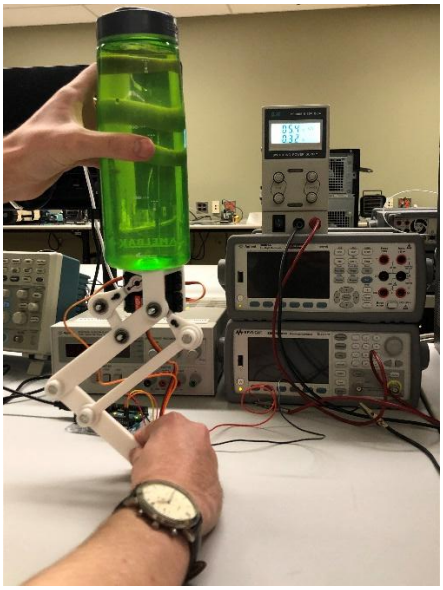


Figure 3-8 Test of the leg supporting weight. The hands are there to keep the system balanced and they provided no other assistance to the leg. The water bottle contained 750ml of water indicating the leg supported 750g of weight, assuming density of water is 1g/ml.

3.2.3 Design of Chassis and Rotation of the Entire Leg

To enable the robot to turn, a third DOF is required. Thus, the design incorporates an additional motor that rotates the entire leg, as in Figure 3-9. Without this motor, the leg would only be able to move in a single plane which prevents the robot from turning. The chassis was also designed at this stage, shown in Figure 3-9. The servo motor driver board mounts to the back of the chassis, and the large rectangular cutout in the middle is where the carrier board slots into. The chassis was designed as a single 3D printed part, which introduced problems as the part took over 40 hours to 3D print, and it had no flat surfaces on its underside, so it had to be printed on a bed of support material, making that surface rough.

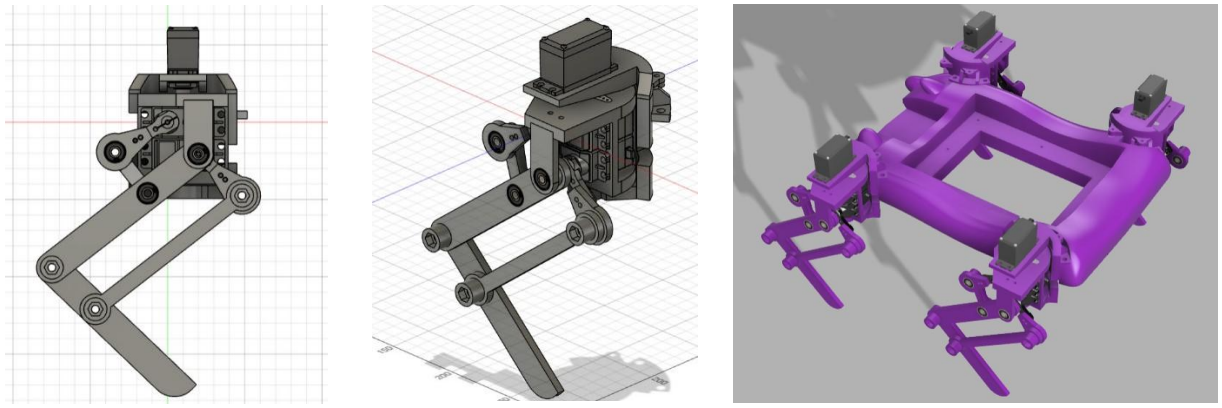


Figure 3-9 CAD models designed for the first walking test. Left to right: front view of the leg, isometric view of the leg, render of the entire quadrupedal robot with its chassis. Each leg has an additional motor on the top which is used to rotate the entire leg for making turns.

3.2.4 A First Attempt at Walking

This design was fabricated, assembled, and first tested to see if the robot could support its own weight while standing as in Figure 3-10, which was successful. To test the robot's walking ability, it was programmed to simply pull back two legs across the ground, while the other two legs provided support, and vice versa. At this stage, we assumed that to generate forward motion, the leg would simply have to push in the opposite direction against the ground, then by Newton's third law, a forward force would be applied on the robot.

Upon testing the walking ability with that assumption in mind, the robot was in fact able to "walk" although its speed was extremely slow and unreliable. The robot would tend to turn rather than walk straight, and when the turning was deliberately tested by rotating all four legs in the desired direction, no change in direction of motion was observed. In addition, the leg was far too flexible as when the robot tried to walk, we observed the leg components bent by approximately 1-2 cm. Finally, the robot's traction appeared to be an issue as it looked like the robot slipped when trying to walk, although future testing revealed this may not have been as much of an issue as originally thought. All these issues were then addressed in the final design iteration, most importantly of which was the walking algorithm, which was upgraded to using inverse kinematics.

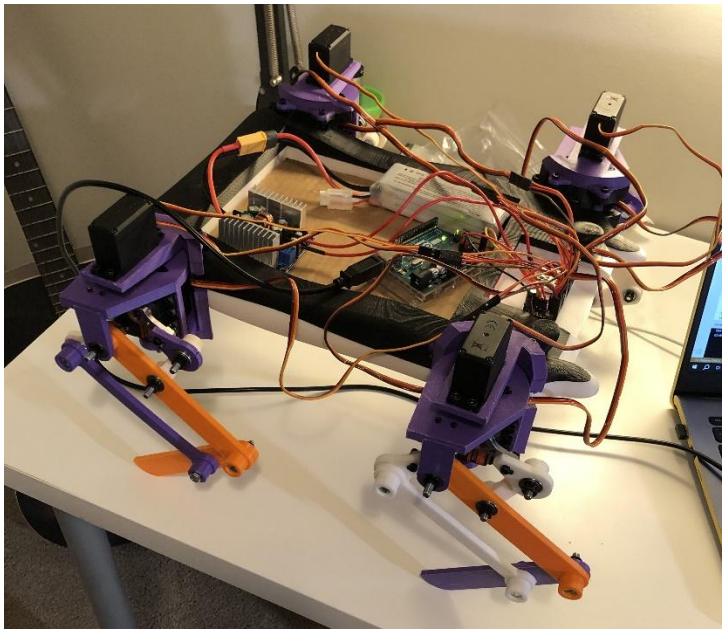


Figure 3-10 First test of the quadrupedal system standing. When the motors are not powered, they don't produce enough torque, and the robot collapses under its own weight. This test verified that the motor were functional and there was enough mechanical strength for the robot to stand.

3.2.5 Design of Leg Structural Components

To address the observed flexibility issues, each component of the leg was made 15 mm thick, opposed to the original 5 mm which was too flexible. **Error! Reference source not found.** showcases the final design of the leg. Additionally, the previous design had each leg component connect by stacking on top of each other, similar to how a pair of scissors is composed of the two blades stacked together. This caused the foot to stick out somewhat rather than be directly below and in line with the mass it was supporting which puts an extra torque on the leg, causing it to bend. To address this, the final components were designed such that the foot and top of the leg remained in line. This structural design for the leg completely resolved all the bending issues we observed during previous testing.

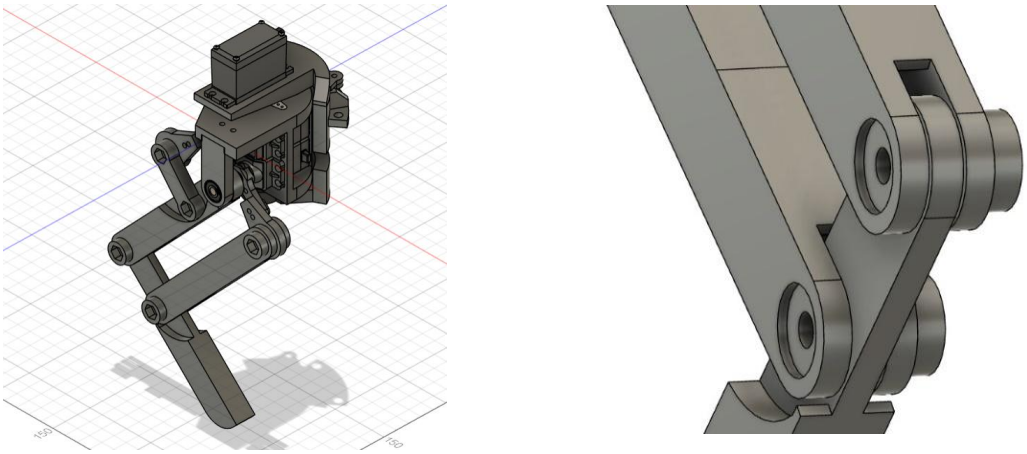


Figure 3-11 CAD model of the final design of the leg. Left: Isometric view of the leg, right: lower joint of the leg, note that the upper lower leg member is connected to both sides of the upper components. Thickness of all components was increased from 5 to 15mm.

3.2.6 Inverse Kinematics

To understand the walking algorithm used by the quadrupedal system, we need to briefly discuss forward and inverse kinematics. Forward kinematics is simply the problem of taking a mechanical system where all the joint configurations are known, then using that information to determine the configuration (position) of the end-effector, which for ServoSentry is the foot. An example would be if one measures the angle a person's arm makes relative to their body, and the angle the lower arm makes relative to the upper arm, the position of their hand in space could then be calculated. The much harder problem is inverse kinematics, where only the location of the hand in space is known, and the geometry of the arm. That information would then be used to go in reverse and determine the configuration of the arm's joints. This is useful for robotics because the user can choose the position that the foot of the quadrupedal robot should be at, and then the required motor output angles automatically get calculated.

Luckily, the inverse kinematics problem for this specific leg geometry is rather straightforward as illustrated in Figure 3-12. To start, a desired location of the foot is chosen relative to the coordinate system (indicated in blue in Figure 3-12). The two leg members highlighted in red in the left image then need their orientation determined, with the key idea being they must remain connected together which becomes the problem of finding the intersection of two circles. The x and y coordinates of their intersection point are given by [34]:

$$(x,y) = \frac{1}{2}(x_1 + x_2, y_1 + y_2) + \frac{C_1}{2}(x_2 - x_1, y_2 - y_1) \pm \frac{1}{2}\sqrt{2C_1 - C_2 - 1}(y_2 - y_1, x_1 - x_2) \quad (3.1)$$

$$where \quad C_1 = \frac{r_1^2 + r_2^2}{R^2}, \quad C_2 = \frac{(r_1^2 - r_2^2)^2}{R^4}$$

Where x_1, y_2, r_1 and x_2, y_2, r_2 are the positions and radii of circles 1 and 2 respectively, and R is the distance between the centers of the circles. This equation has two solutions as circles can intersect at two points, but since we know the geometry of the leg, we can always select the correct solution. The position of two of the leg members is now determined, and the process repeats for the remaining linkages (right picture in Figure 3-12) until the positions of the two topmost leg members are known, which can then be used directly to calculate the angles the motors must be set to.

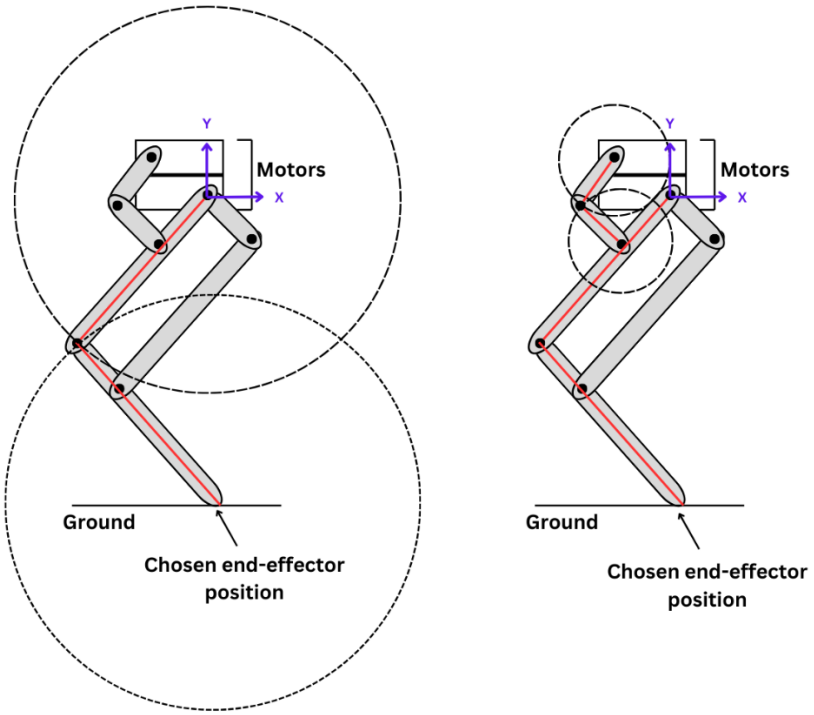


Figure 3-12 Inverse kinematics. XY coordinate axis are indicated in blue, and each filled black circle represents a joint of the leg that can rotate. The goal of inverse kinematics is to choose a position of the end-effector (the robot's foot) and from that calculate the joint configuration which produces that position. Left: using the foot position, the position of the leg members in red are determined by intersection of circles. Right: using the same procedure, the next members highlighted in red are calculated, this yields the required angle for the upper motor.

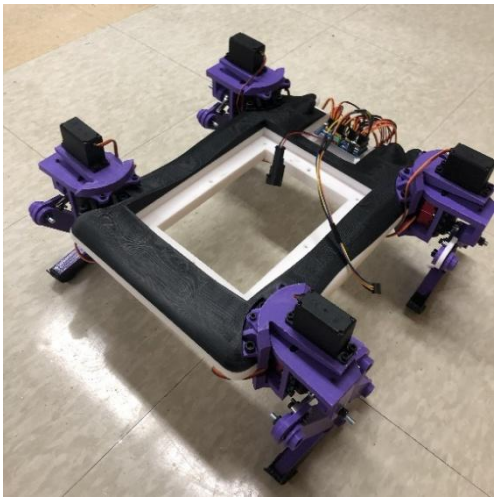
3.2.7 Walking Algorithm

Inverse kinematics is the primary component of the walking algorithm because it allows us to easily set desired location of the foot quickly and with greater accuracy over the previous method of trying to guess which joint configurations would produce desired motion. The inverse kinematics algorithm was very accurate when compared against the CAD model in Fusion 360, and there was an approximate 5 mm difference between desired and actual positions when tested with the fabricated leg. This error can likely be attributed to the play in the mechanical joints, and the error between the desired and actual motor angles when trying to control them.

Built on top of the inverse kinematics, is the walking pattern used by ServoSentry. We assumed in previous walking tests, that pushing backwards against the ground would propel the robot forwards, however that ultimately did not work. Instead, we tried to base the walking pattern on how a person typically walks, as a person would step forwards with one leg, then shift their body weight onto that leg, and finally bring their other leg forwards. The final walking pattern mimics this concept of shifting its weight to move, and it proved successful. The robot would push the entire chassis forward, while keeping the feet at the same location on the ground, then each pair of diagonally opposite legs would step forwards, causing the entire robot to be translated forwards. Finally turning the robot left and right was achieved by moving the legs on the left or right side while keep the other side fixed as a pivot point. The robot no longer slipped with this approach to walking, indicating that the previous observations of slipping were not mechanical in nature but instead software based.

3.2.8 Final Outcome

The final design for the quadrupedal system achieved all our goals except for attaining a movement speed of at least 25 cm/s. We measured the robot's average speed by recording it from above and measuring a change in distance and how long that distance change took. We found that the average speed of the robot was only 1.63 cm/s which is nowhere close to our performance metric of 25 cm/s. Nonetheless we still consider this system to be successful, as that performance metric was set somewhat arbitrarily as it was difficult to estimate a realistic speed without prior experience of what is achievable with this type of robot. The leg and motor assembly were able to support a total weight 1.75 kg with 0.75 kg being the chassis weight. This result meets our performance metric goal of supporting 1.75 kg total. The completed quadrupedal system is shown in Figure 3-13.



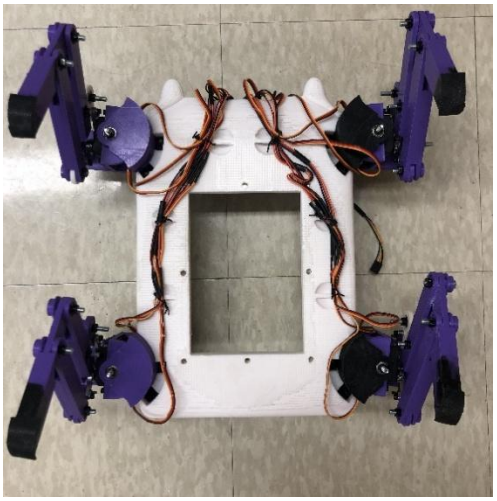


Figure 3-13 Completed quadrupedal system. Left: Isometric view of the robot. The two loose cables are the power and signal connections to the carrier board. Right: Underside view of the robot.

3.2.9 Integration and Programming Interface

To integrate the quadrupedal system with the rest of ServoSentry, a small, standalone Python package named `legControl` was written. This package handles all the quadrupedal system's movement capabilities while exposing to the rest of the codebase a simple to use public interface. The package is used simply by creating an instance of `class LegControl` which takes no parameters in its constructor and provides the public interface. Table 3-1 provides a description of the programming interface for the quadrupedal system. The functions `LegControl.connect()` and `LegControl.disconnect()` enables the program to lose or establish a connection to the servo driver board without throwing any errors. This functionality is required to swap locomotion systems. Whenever a function prefixed with `animation` is called, the robot loops that animation (walking pattern) until the animation is changed again.

Table 3-1: Description of the interface provided by the LegControl class.

Function	**Description**
`LegControl.connect()`	Establishes a connection with the servo driver board.
`LegControl.isConnected()`	Returns a Boolean indicating connection status.
`LegControl.disconnect()`	Disconnects the servo driver board.
`LegControl.animationStand()`	Begin playing the standing animation.
`LegControl.animationWalk()`	Begin playing the walking animation.
`LegControl.animationTurnLeft()`	Begin the turning left animation.
`LegControl.animationTurnRight()`	Begin the turning right animation.

`LegControl.update()`	Called in the program's main loop to cycle through the robot's current animation.

3.3 Carrier Board

The carrier board was designed to be a container that would hold the electronics for ServoSentry and be able to be easily changed between the wheeled and quadrupedal locomotion systems by using magnets embedded in the board and chassis as well as plugs to be able to easy connect and disconnect the wires that need to go to the chassis. It was designed to be able to securely hold all the components to ensure nothing would fall out and be damaged in the case of rough terrain. The performance metric for interchange speed was less than 15 seconds and the actual interchange speed ended up being 25 seconds.

The carrier board was designed to hold the primary control boards down by bolting them to the main board. This included the Jetson Nano, the battery and one of the DCDC converters. Furthermore, the Raspberry Pi was placed on a shelf on the main board along with two holders for the Wi-Fi antenna and mounting points for the subassembly that held the remaining electronics were included in the design. Furthermore, holes in the bottom of the board for the magnets were added that would allow for the carrier board to easily transition from one chassis to the other. The magnets are embedded on each locomotion chassis and the underside of the carrier board to secure the two systems together as demonstrated in Figure 3-14.

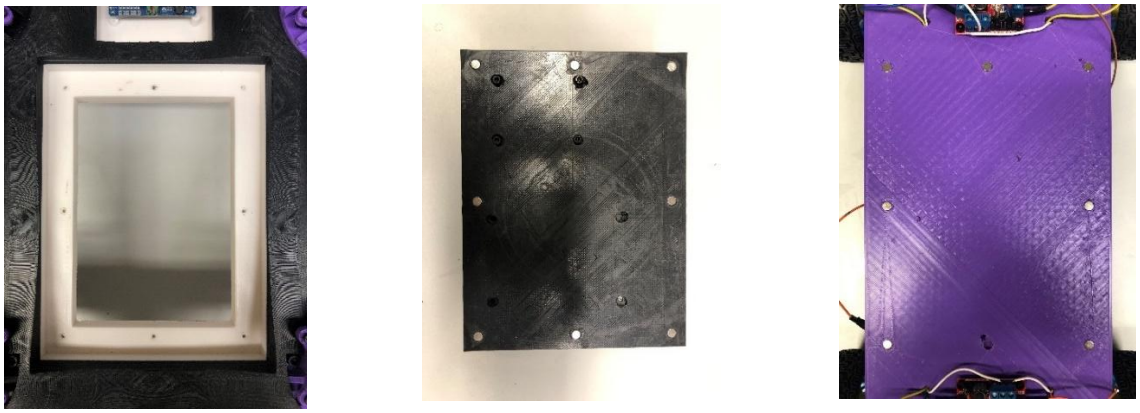


Figure 3-14 Locations of the magnets in each module for securing the carrier board to the locomotion system. Note that for the quadrupedal system the magnets are inserted from the other side of the chassis.

Sitting on top of the main carrier board as shown in the final design in Figure 3-15, a subassembly was added that primarily serves to mount the second DCDC converter, ultrasonic sensor, USB web camera and Raspberry Pi camera. The subassembly also houses an interface

board which allows for easily interchanging the wires that need to be swapped between each chassis. Finally, the subassembly increases the cameras field of view as can be seen in Figure 3-16. This increased field of view allows for the computer vision portion of the project to more easily spot intruders as is outlined in the computer vision section below.

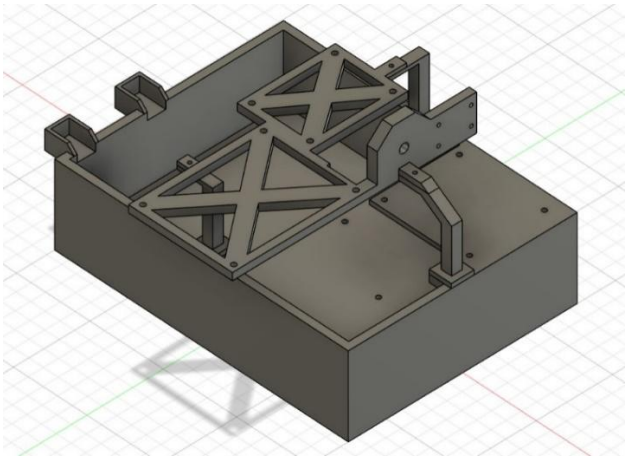


Figure 3-15 Final CAD model of the carrier board with the subassembly sitting on top.

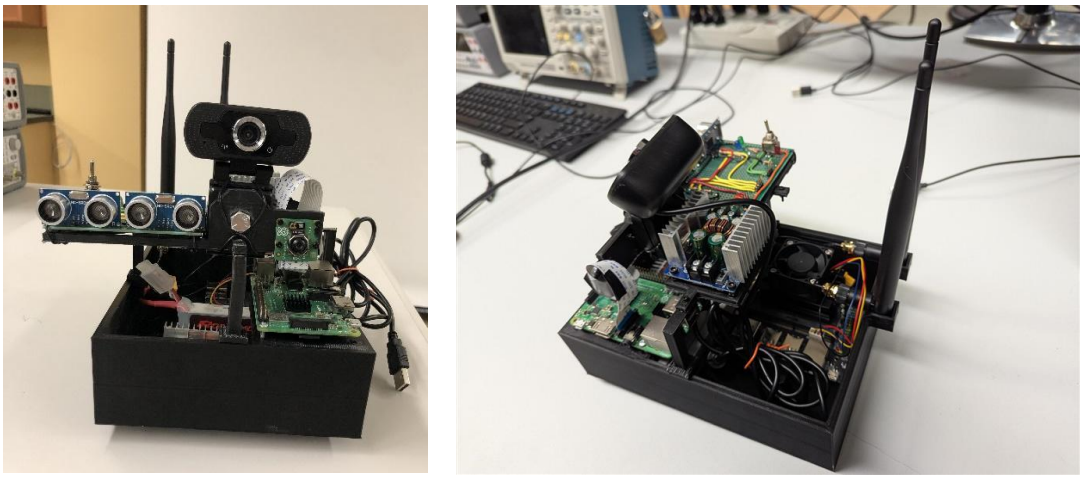


Figure 3-16 Front-facing and isometric views of the assembled carrier board.

4 Computer Vision System

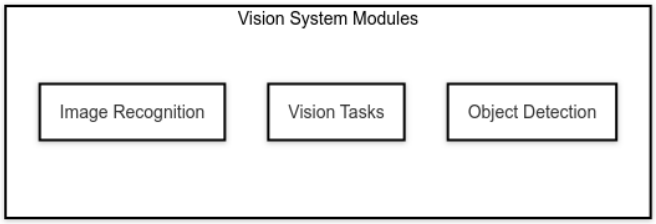


Figure 4-1 Block Diagram Representing the Vision System Modules.

ServoSentry's computer vision system is designed to provide real-time object detection, masked intruder identification, and face recognition as a stretch goal, using deep learning models. As shown in Figure 4-1, the vision system comprises of 3 main modules: Image Recognition, Vision Tasks and Object Detection. The system operates on the NVIDIA Jetson Nano, leveraging its GPU-accelerated CUDA cores to process video feeds from a connected USB camera in real time. YOLOv3 has been utilized for general object detection and a masked-person detection model, ServoSentry can identify intruders and assist in autonomous navigation.

The vision system is responsible for:

- Detecting and tracking individuals in real time.
- Identifying masked persons as potential intruders.
- Providing object classification for environmental awareness.
- Assisting the robot in making navigation decisions based on bounding box data.

The following sections provide contexts on core components, models used, and implementation strategies for ServoSentry's vision-based intelligence.

4.1 Why Masked-Person Detection?

A key element of our demonstration involves detecting an individual wearing an unusual face covering, such as a surgical mask, scarf, or other unconventional item over the face. While this might initially sound security-focused, e.g., identifying an "intruder" in a restricted area; it is a convenient test scenario for our vision pipeline.

1. **Clear Differentiation:** It is straightforward to train a neural network to distinguish between a "normal" face and a "covered" face. This clear contrast allows us to validate whether our detection pipeline can reliably classify two visually distinct categories.
2. **Simple Testing:** Using an everyday surgical or cloth mask is an easy, repeatable way to see if the robot can detect and follow a person once it recognizes them as "masked." This approach proves that our system can handle real-time object classification and drive the robot accordingly.
3. **Generalizable Approach:** Although we often refer to "masked person detection," the same method can be adapted to detect any face covering or suspicious appearance. The model can be retrained to spot various attributes (e.g., hats, helmets, or entirely different objects) with minimal changes to the pipeline.

Thus, "masked-person detection" serves as a practical demonstration of how our computer vision module can identify a user-defined category (in this case, a face covering) and trigger the robot's behaviors, such as alerting, following, or avoiding. It is not a limitation of the system; rather, it showcases how quickly we can tailor the robot's recognition tasks to new or unconventional appearances.

4.2 Object Detection

4.2.1 YOLOv3: Real-Time Object Detection

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection system optimized for speed and accuracy. Unlike traditional region proposal-based networks (e.g., Faster R-CNN), YOLOv3 processes an entire image in a single forward pass, significantly reducing computation time while maintaining high detection precision [4].

The YOLOv3 model is structured as follows:

- **Darknet-53 Backbone:** A convolutional neural network comprising 53 layers with residual connections, allowing for efficient feature extraction.
- **Multi-scale Prediction:** Objects are detected at three different scales, ensuring robustness across various object sizes.

- **Bounding Box Regression:** Predictions are refined based on predefined anchor boxes and a grid-based localization approach.
- **Object Classification:** A final detection layer assigns class probabilities to identified objects using logistic regression [4].

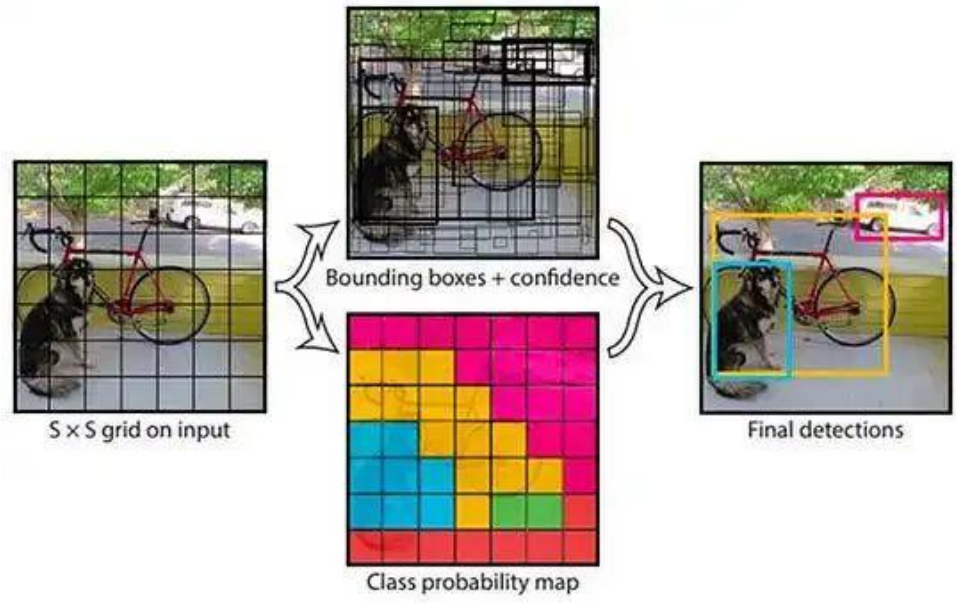


Figure 4-2: YOLOv3 Architecture Diagram showing how YOLOv3 splits the image into an S×S grid, with each grid cell predicting multiple bounding boxes and class probabilities [35]

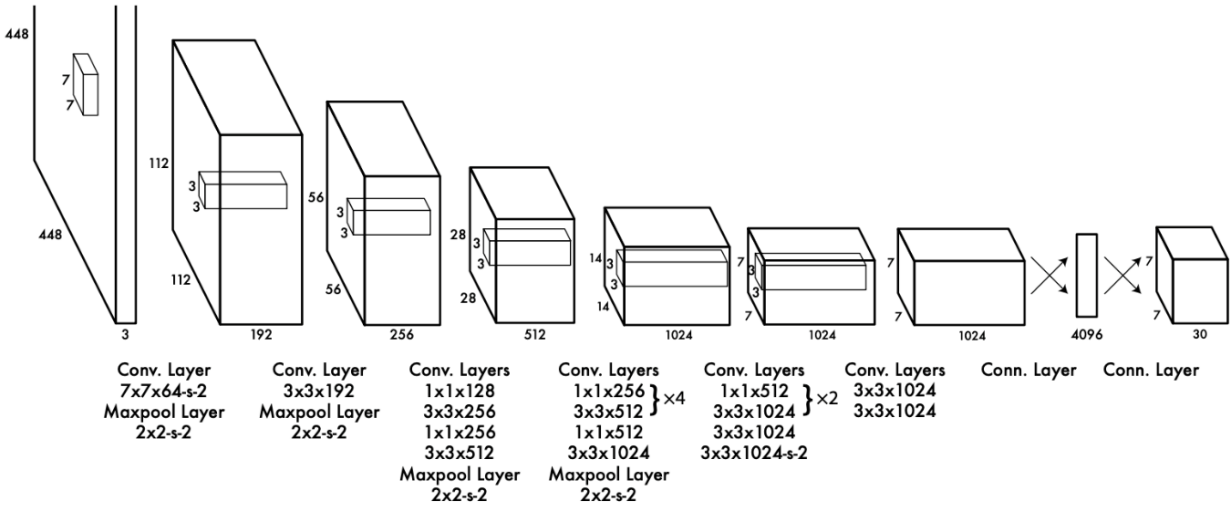


Figure 4-3: YOLO detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1x1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224x224 input image) and then double the resolution for detection [4].

4.2.2 Object Detection Pipeline in ServoSentry

The YOLOv3 detection process begins with image preprocessing, where the input image is resized to a 416×416 resolution and its pixel values are normalized to ensure consistency with the model's architecture. Once prepared, the image is passed through Darknet-53, a deep convolutional network that extracts feature maps at multiple scales, capturing fine details as well as broader structures.

To localize objects, YOLOv3 divides the image into a grid system, where each cell is responsible for predicting objects, whose center falls within it. The model generates bounding box coordinates along with confidence scores and class probabilities, allowing it to determine both the presence and category of detected objects.

Since multiple bounding boxes may overlap for the same object, Non-Maximum Suppression (NMS) is applied to refine the predictions. This step eliminates redundant detections by selecting only the most confident bounding box for each object based on Intersection over Union (IoU) thresholding.

After filtering, the final output consists of precise bounding boxes with their corresponding labels, enabling real-time object tracking and recognition [4].

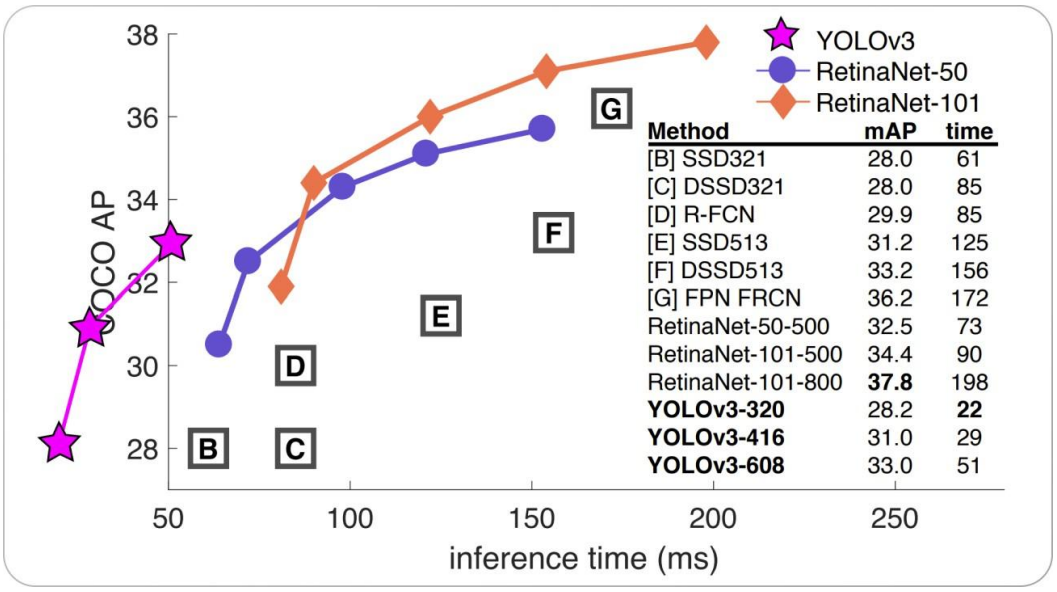


Figure 4-4: Speed-accuracy trade-off for YOLOv3, RetinaNet, and other object detectors on the COCO dataset [35].

In Figure 4-4, the y-axis shows average precision (AP), while the x-axis indicates inference time (milliseconds). YOLOv3 demonstrates a favorable balance between high detection performance and real-time inference speeds compared to competing methods [35].

YOLOv3-tiny is chosen for its balance of speed and accuracy. The Jetson Nano achieves approximately 10-15 FPS when running YOLOv3 with FP16 precision, which is sufficient for our real-time security application. Further optimizations, such as TensorRT acceleration, may be explored to enhance processing efficiency [4]. Figure 4-5 below demonstrates the utilization darknet's real-time object detection algorithm via a webcam. The screenshot also shows the frame rate achieved and the detection accuracy.

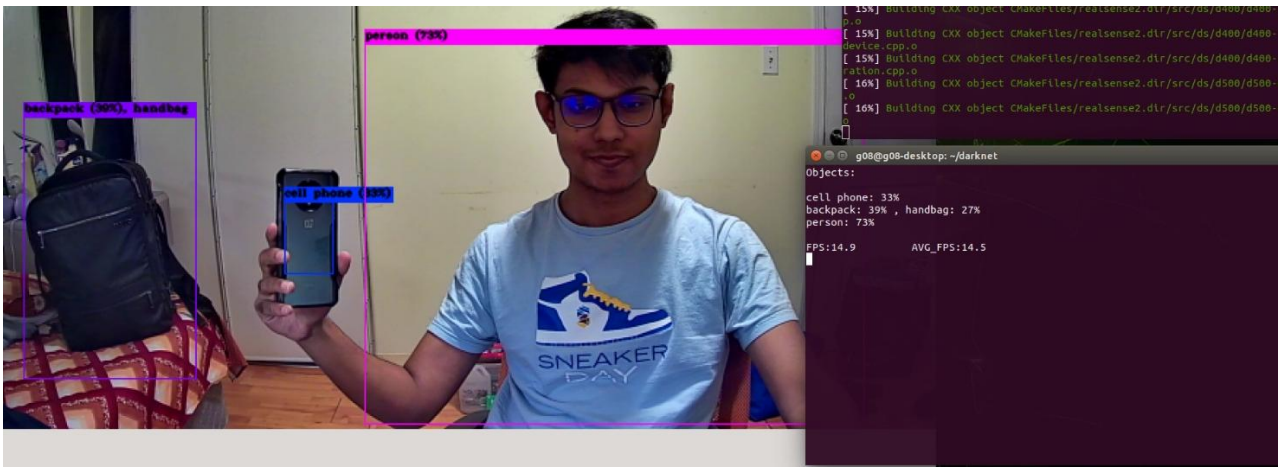


Figure 4-5: YOLOv3 detecting objects while showing detection accuracy and frame rate on our Jetson Nano

4.3 Intruder Detection

ServoSentry's security capability is enhanced by a masked intruder detection model, which builds upon YOLOv3. By integrating a Cansik YOLO Mask Detection repository [36] and combined it with the high-performance Darknet framework maintained by AlexeyAB. As explained earlier, we have defined an intruder as a masked individual. Later in the integration section we have demonstrated and executed a more practical use case of this model.

The first step in this integration involved cloning the Cansik repository and setting up the required environment. The repository provides pre-trained weight files and configuration scripts specifically designed for classifying individuals as wearing a mask or not. Since our application required detecting masked individuals as potential intruders, we modified the class labels in the

dataset replacing "mask" and "no mask" with "intruder" and "person," respectively. This change ensures that the robot prioritizes responses to masked individuals while still being able to recognize and track unmasked persons as general objects within its environment.

To execute real-time inference efficiently, we compiled and configured the Darknet framework on the Jetson Nano, ensuring compatibility with CUDA acceleration. The Darknet repository provides extensive build instructions that we followed, incorporating optimizations such as FP16 precision to enhance inference speed. While YOLOv3 alone is sufficient for general object detection, the additional mask detection model enables ServoSentry to make more precise security-related decisions [37].

During real-time operation, each frame captured by the Jetson Nano is processed concurrently by both detection pipelines. If the mask detection module identifies a masked face, that detection is prioritized, and the corresponding bounding box data is immediately transmitted via UART to the Raspberry Pi. This ensures that the system reacts promptly to potential intruders. If no masked individual is detected, the system defaults to the general YOLOv3 detections, which are used for tasks such as obstacle avoidance and navigation.

4.3.1 Detection Pipeline

The intruder detection module operates in parallel with YOLOv3. Figure 4-6 on the left shows the output when only a person is being detected. While Figure 4-7 on the right shows when both a person and an intruder are in the frame. The comparisons are as follows:

- YOLOv3 detects multiple objects in the frame that consist of people. Figure 4-6 shows 2 models working simultaneously to detect a person only.

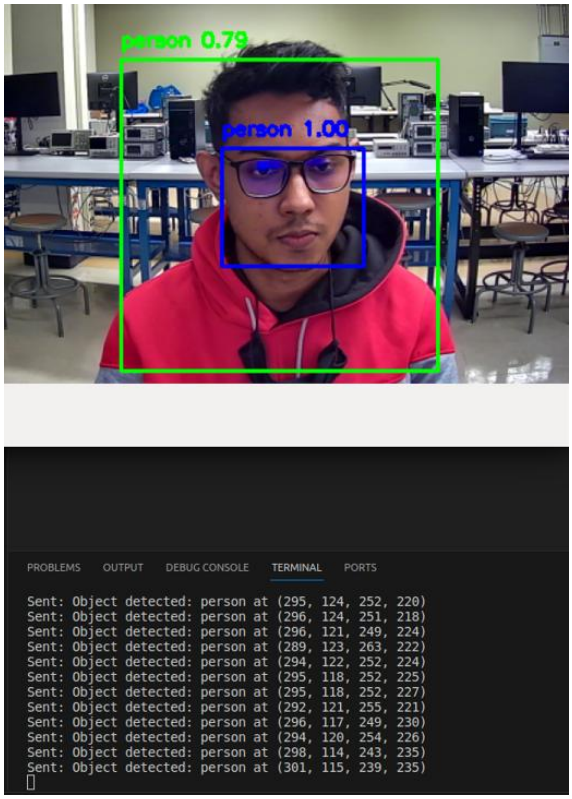


Figure 4-6: Vision System Detecting People using both the masked model and YOLOv3 COCO dataset.

- The mask detection model below focuses solely on identifying faces and determining whether they are covered with a face covering. Figure 4-7 shows the model detecting a masked person as "intruder" class while other people are still detected as "person" class by the same model.

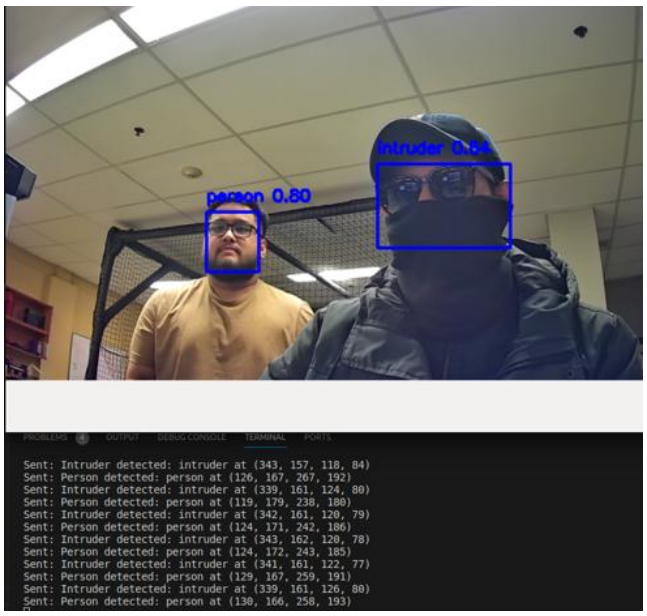


Figure 4-7: Only the intruder detection model being implemented when a face covered individual is in the frame.

Appendix-A, Figure 0-1 and Figure 0-2, illustrates the training progression of the YOLOv3 model, and the loss mAP curves for the mask detection model during training where the loss decreases steadily, and the mean average precision (mAP) improves over time.

An essential part of this process was ensuring that the mask detection model was well-trained and optimized for real-time deployment. Figure 0-1 illustrates the loss curve from the YOLOv3 training process, showing how the model progressively improves in accuracy over multiple iterations [37].

Table 4-1: Key Performance Parameters of Computer Vision Models

Parameter	**YOLOv3 Model**	**Mask Detection Module**
Inference Time	~0.067 sec per frame (~14.8 FPS)	Comparable to YOLOv3; estimated within 10–15 FPS when optimized
Detection Accuracy	~70% for intruders; ~70% for general person detections; ~55% for other objects [12]	Convergence achieved over 6000 iterations; mAP improves steadily
Backbone / Architecture	Darknet-53 with residual connections	Adapted from Cansik's implementation with modifications for relabeling
Optimization Techniques	FP16 precision; potential TensorRT acceleration	Similar optimizations applied; uses Darknet framework for real-time inference
Training Dataset	COCO dataset for general objects	Custom dataset derived from the mask detection repository (modified labels: "intruder" and "person")

4.4 Bounding Box Data Processing

After the object detection and mask detection pipelines generate their respective outputs, the bounding box processing module refines these results to produce precise, actionable data for navigation. When YOLOv3 processes an image, it divides the image into an S×S grid, and each cell predicts multiple bounding boxes along with associated confidence scores and class probabilities. These predictions are provided in a normalized format, where the coordinates typically representing the center (x, y) and the dimensions (w, h) of each box are expressed as ratios relative to the image dimensions.

Because overlapping predictions are common, multiple boxes may be generated for a single object. To address this redundancy, our system implements Non-Maximum Suppression (NMS). NMS works by first selecting the bounding box with the highest confidence score and then eliminating any boxes that have a high Intersection over Union (IoU) overlap (i.e., exceed a preset threshold) with the selected box. This iterative process ensures that only the most reliable detection remains for each object, thereby reducing false positives and simplifying subsequent processing [38].

After applying NMS, the normalized coordinates are converted to absolute pixel values based on the original image dimensions. This conversion is crucial for accurately mapping detected objects in the camera frame to real-world positions, which are then used by the navigation algorithms [35].

Figure 4-8 shows a processed image demonstrating Non-Maximum Suppression applied to overlapping bounding boxes, with only intruder data transmitted via UART to the raspberry pi. This can be confirmed from the received coordinates seen on the second screenshot in Figure 4-9 as the raspberry pi receives only the intruder coordinates.

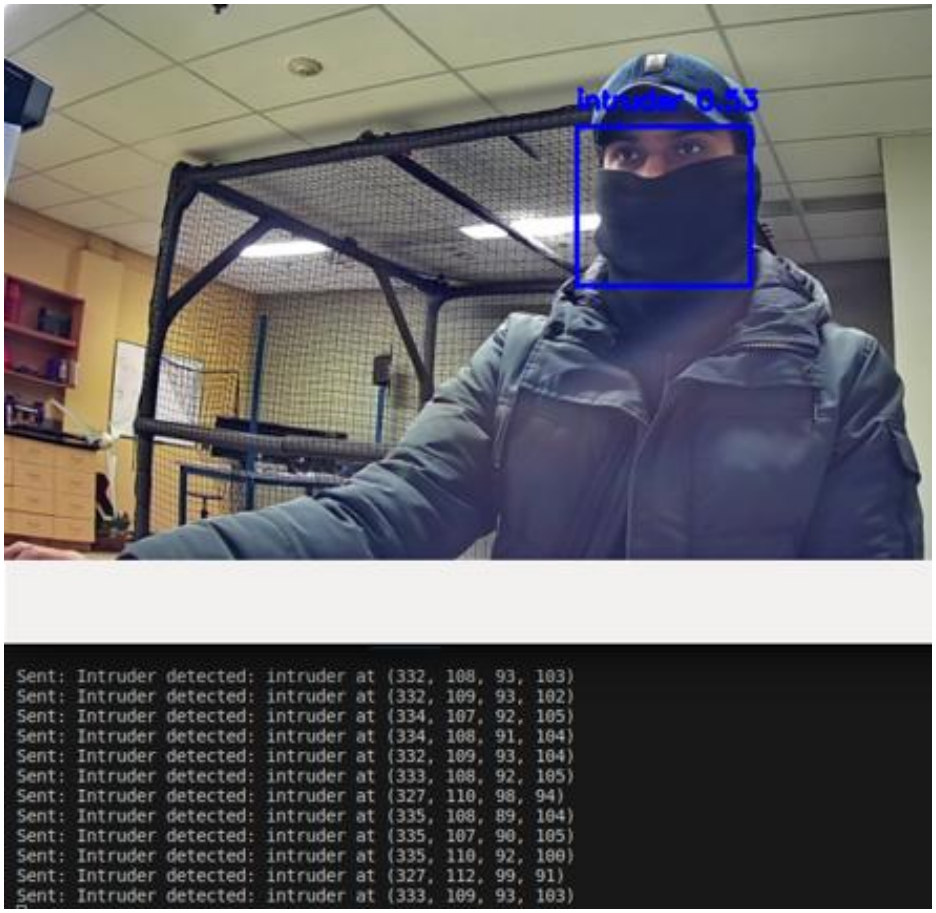


Figure 4-8: Screenshot showing an intruder and a non-intruder detected by the system. The image displays processed bounding boxes with absolute coordinates (X, Y, w, h), demonstrating the effect of Non-Maximum Suppression and confidence thresholding.

In security-critical situations, if the mask detection module identifies an intruder, the corresponding bounding box data is transmitted immediately via UART to the Raspberry Pi. This ensures that intruder data is prioritized over other detections. The refined bounding box dimensions, in conjunction with real-time distance measurements from the ultrasonic sensors,

enable the robot to make informed navigation decisions such as adjusting its path to avoid obstacles or accurately following a detected intruder until a predefined proximity threshold (approximately 20 cm) is reached.

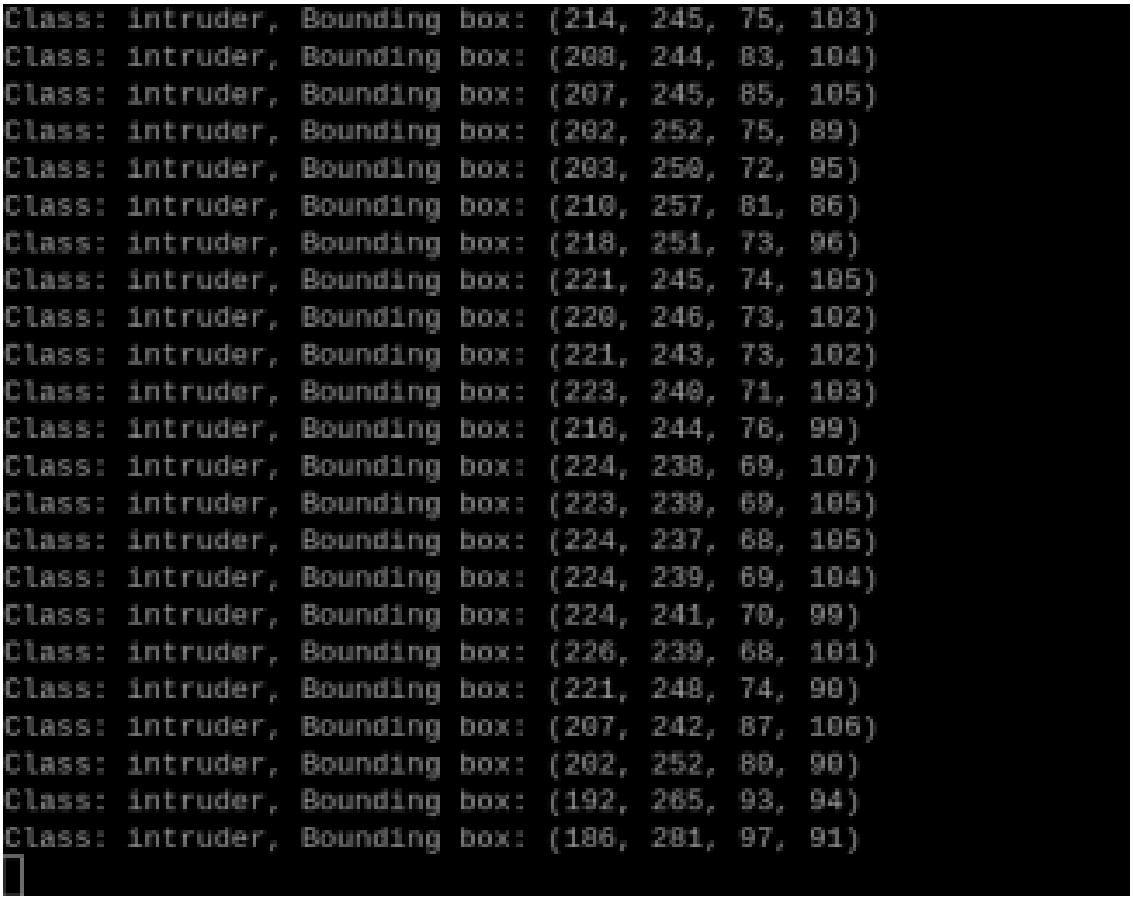


Figure 4-9: Screenshot of the Raspberry Pi terminal receiving the bounding box intruder data sent from the Jetson, using UART.

4.4.1 Remote Access

The data retrieved from the computer vision system is sent to administrators via the authenticated live webcam accessible over the network. Security personnel can view the live feed to determine the severity of the incident and access visual data otherwise unavailable. Manual directional input is also available to move ServoSentry to specific locations.

4.5 Facial Recognition

The current implementation uses masked detection as a placeholder and that the team is developing a more refined face recognition module to address the high prevalence of masks in certain environments. This is why the face recognition pipeline isn't fully operational on the Jetson Nano.

4.5.1 Face Recognition Model

Face recognition is an advanced feature incorporated into ServoSentry to enhance its security and identification capabilities. This module allows the system to distinguish between authorized individuals and potential intruders using deep learning-based facial recognition. The implementation is built upon the facenet-pytorch framework, leveraging Multi-task Cascaded Convolutional Networks (MTCNN) for face detection and InceptionResnetV1 for feature extraction and classification. [30]

MTCNN (Multi-Task Cascaded Convolutional Networks) is a robust and precise algorithm for face detection that outperforms traditional methods like Haar Cascades. It is particularly effective in challenging conditions with varying face sizes, orientations, and lighting. The process is divided into three distinct stages:

- **Proposal Generation:** In the initial phase, a lightweight neural network scans the input image and generates a set of candidate regions (bounding boxes) that may contain faces.
- **Refinement:** In the second phase, a subsequent neural network reviews these candidate regions, fine-tuning the bounding boxes to more accurately match the actual face boundaries. This step improves overall detection accuracy.
- **Facial Landmark Detection:** Finally, a third network identifies key facial landmarks, such as the corners of the eyes, the nose, and the mouth; ensuring that each face is precisely localized.

This cascaded approach allows MTCNN to achieve high detection performance even under diverse and challenging conditions [39].

Appendix-A, Figure 0-3, illustrates the MTCNN cascaded pipeline, which comprises three stages of multi-task deep convolutional networks. First, a fast Proposal Network (P-Net) generates candidate windows. Next, these candidates are refined by a Refinement Network (R-Net). Finally, an Output Network (O-Net) delivers the final bounding box coordinates and the positions of facial landmarks [40].

4.5.2 Model Selection and Training Process

The face recognition pipeline consists of two primary phases: training and real-time inference.

1. **Face Detection:** The MTCNN model is responsible for identifying faces in each image or video frame. It detects facial landmarks and extracts cropped regions containing faces [30].
2. **Feature Extraction:** The cropped faces are passed through InceptionResnetV1, a deep convolutional neural network (CNN) trained on large-scale facial datasets such as VGGFace2. This model generates fixed-length embeddings that uniquely represent each face [30].
3. **Classification:** The embeddings are normalized and fed into a Support Vector Classifier (SVC), which is trained to differentiate between known and unknown individuals [30].

To ensure the model correctly identified authorized individuals, we collected a dataset consisting of:

- **Positive samples:** Images of the authorized user labeled as "Kazi."
- **Negative samples:** Images of various other people labeled as "Not Kazi."

The training process involved extracting embeddings from both datasets, normalizing them, and training the SVC model. This classifier was saved as a .pkl file using joblib, allowing it to be loaded without retraining during real-time execution [41].

4.5.3 Implementation and Real-Time Detection

During testing on a high-performance desktop equipped with an NVIDIA RTX 4060 GPU, the model successfully achieved real-time recognition with high accuracy, consistently identifying the authorized user even with variations in lighting and facial angles. The system displayed bounding boxes around detected faces and labeled them accordingly. Figure 4-10 below shows the model successfully detecting the positive samples of a teammate’s picture provided against a negative sample which does not necessarily have to be in the dataset.

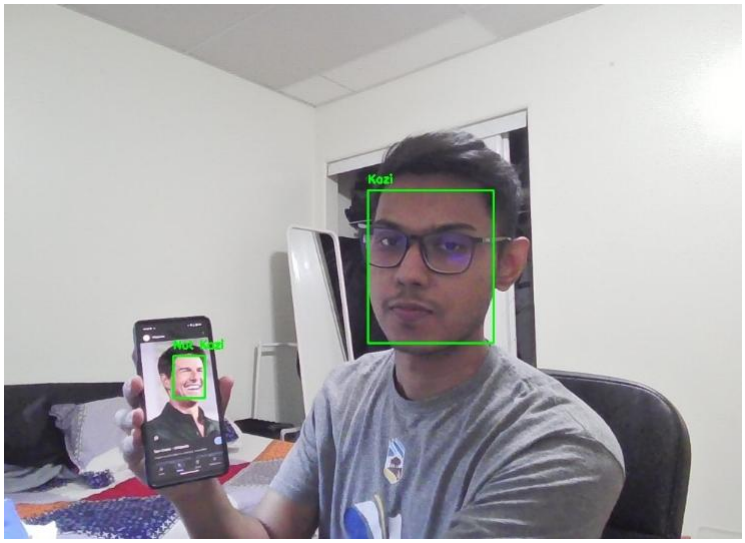


Figure 4-10: Face recognition model successfully identifying an authorized user versus an unknown individual.

4.5.4 Challenges with Integration

While the system performs well on a high-end desktop, running it on the Jetson Nano has presented several challenges due to computational and compatibility limitations:

1. **Dependency Issues:** The Jetson Nano uses an ARM-based architecture, which caused incompatibility issues with NumPy, Matplotlib, and Joblib. The prebuilt NumPy wheel failed to import the `_core` module, requiring a manual rebuild using `--no-binary :all:` [5].
2. **Performance Bottlenecks:** Unlike the desktop setup, where the model runs on a high-speed CUDA-enabled GPU, the Jetson Nano's limited processing power significantly slows down face recognition [42].
3. **Serialization Problems:** The .pth format originally used for model storage caused errors during loading. Converting the model to a .pkl format using joblib helped resolve this issue, but execution speed remained a concern [41].
4. **Thermal Constraints:** Due to the high computational load, the Jetson Nano experienced thermal throttling, requiring the use of a cooling fan configured with the Pyrestone jetson-fan-ctl script [5].

Despite these challenges, we have successfully executed the MTCNN face detection module on the Jetson Nano. However, integrating the full-face recognition pipeline in real-time remains problematic. Future optimizations such as TensorRT acceleration or an external processing unit could significantly enhance performance [5].

5 Communication System

The communication system plays an important role in ensuring data exchange between the Jetson Nano and the Raspberry Pi, as well as allowing for remote access for monitoring and control. The system must allow for Wi-Fi connectivity with a host system and should maintain connectivity with the host system for a minimum distance of 1m.

5.1 Jetson Nano to Raspberry Pi Communication

The Universal Asynchronous Receiver/Transmitter Protocol (UART Protocol) was used to establish communication between the Jetson Nano and Raspberry Pi. The Jetson Nano transmits the boundary box coordinates of detected objects to the Raspberry Pi in real time.

As outlined in Chapter 4, the YOLO object detection model is implemented on the Jetson Nano to analyze a live video stream and identify the target of interest: a person wearing a mask. It sets a boundary box on the target and assigns coordinates which represent the target's position. Using UART, the boundary box coordinates are transmitted to the Raspberry Pi. The Raspberry Pi, which is the receiver processes the coordinates to allow ServoSentry know how close the target is to it.

UART is a widely used protocol that allows for data transmission between devices serially. It uses two wires (TX and RX) to send and receive data one bit at a time asynchronously. UART was chosen due to its simplicity in hardware connection. Also, the transmit and receive pins on both the Jetson Nano and the Raspberry Pi use 3.3V so we did not have to use voltage level shifters. To setup the connection between the two devices, we:

- Connected the Jetson Nano's UART TX (Transmit) Pin to the Raspberry Pi's UART RX (Receive) Pin.
- Connected the Jetson Nano's UART RX (Receive) Pin to the Raspberry Pi's UART TX (Transmit) Pin.
- Connected both devices to a common ground.

Figure 5-1 below shows a visual representation of the connection:

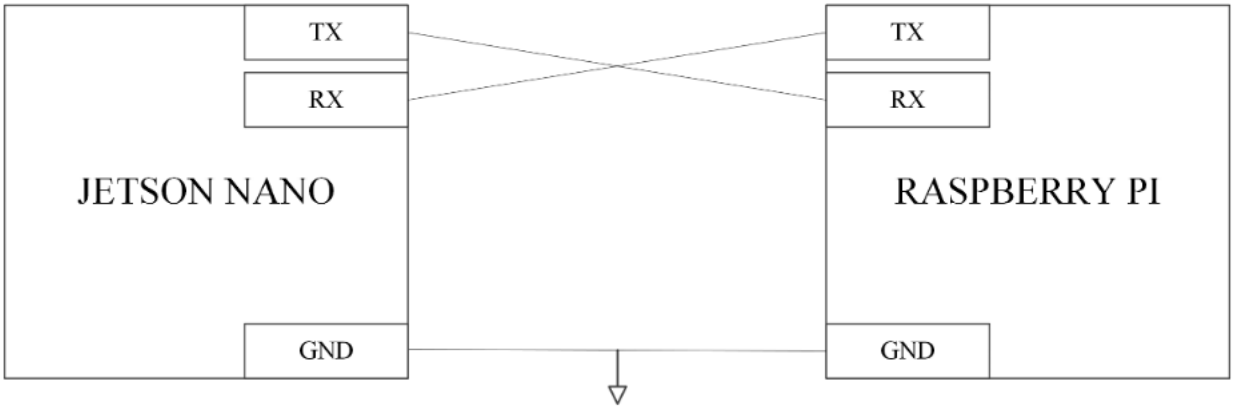


Figure 5-1: Block diagram showing UART wiring.

5.1.1 UART Configuration

The baud rate on both Jetson Nano and Raspberry Pi to were set to 115200. This baud rate allowed for faster data transfer between the two devices, ensuring the timely delivery of bounding box coordinates while maintaining stable communication. Since Python was used, parity bits, start and stop bits within the UART data frame were not explicitly set. The Python 'serial' library handles these configurations.

5.2 Wi-Fi Module and Networking

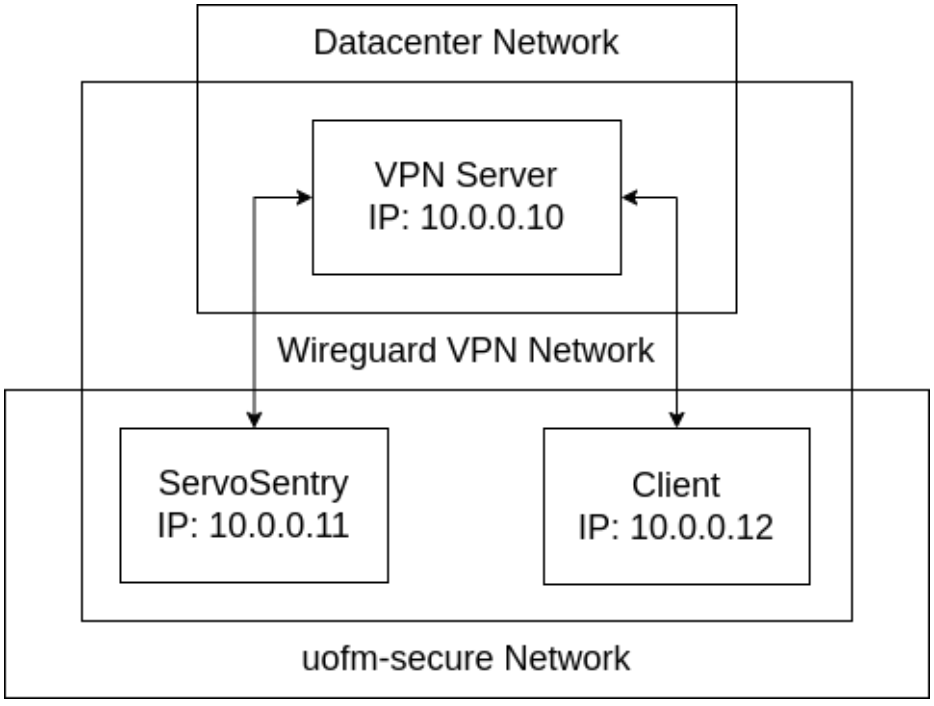


Figure 5-2: VPN Network

To enable remote interaction with the Jetson platform, an integrated M.2 Wi-Fi adapter is used. By default, using Ubuntu's compatibility with `NetworkManager`, ServoSentry connects to both the campus `uofm-secure` network as well as a WireGuard virtual private network (VPN) hosted in the cloud. The VPN server is configured to create a virtual local area network (VLAN) between all connected devices. By using this method, ServoSentry can be defined a static IP address behind secure authentication. This allows remote access by any software or user connected to the same authenticated VPN server regardless of which network ServoSentry is connected to. This is a necessity since static IP addresses are unavailable on the `uofm-secure` network.

As a backup, if there is no internet access in the vicinity, `NetworkManager` is configured to create a wireless access point, allowing nearby devices to establish a connection. The `nmcli` utility is used to configure the `wlan0` interface, generating a secure hotspot with a specified SSID and password. Network settings are configured to implement a shared connection and to enforce standard security protocols. The auto connect feature ensures the hotspot is readily available upon system startup, with a lower priority than the `uofm-secure` nework.

A secure shell (SSH) server is configured on ServoSentry for command line access to execute programs, and a virtual network computing (VNC) server is also configured allowing a remote graphical user interface for the Ubuntu desktop environment. Users can connect to these servers through a locally installed SSH or VNC client. Once connected to either of these two servers, the real-time camera live feed can be viewed, and manual directional input can drive ServoSentry's movements.

6 Sensors

Two HC-SR04 ultrasonic sensors are used in ServoSentry to provide real-time distance measurement and obstacle avoidance capabilities. Their simple interfacing requirements and widespread documentation allowed us to rapidly develop and integrate them into the control system with minimal design overhead. Ultrasonic sensors provide accurate time-of-flight measurements, making them well-suited for real-time applications where swift obstacle detection is crucial for autonomous navigation. These sensors work by emitting ultrasonic pulses and measuring the time it takes for the pulses to reflect off objects and return to the sensor. The measured time is converted to distance using the speed of sound as a reference.

Due to the voltage differences between the Jetson Nano and the HC-SR04 sensor, a safe interfacing method was required. We directly interfaced the sensors with the Raspberry Pi using a voltage divider circuit demonstrated in Figure 6-1. The circuit ensured that the Echo pin output was safely reduced from 5V to 3.3V, preventing potential damage to the Pi's GPIO.

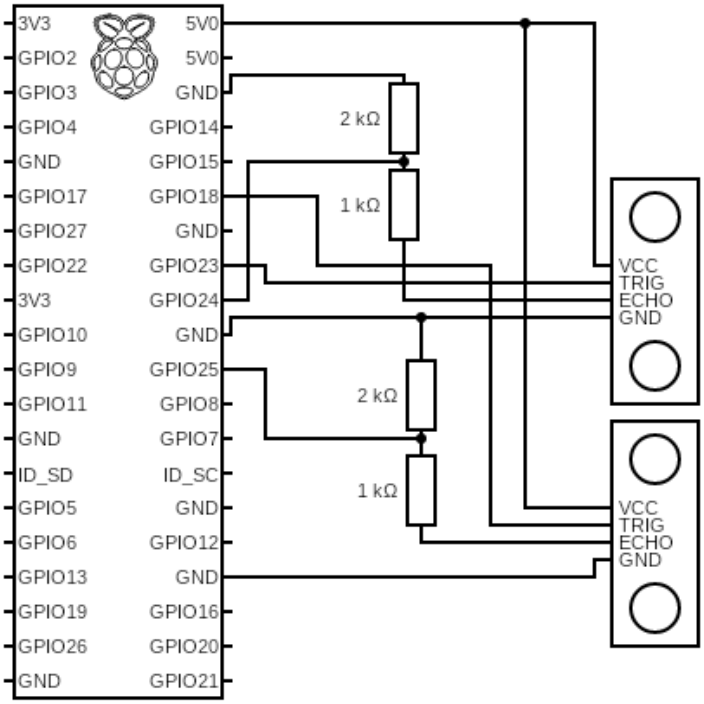


Figure 6-1: Ultrasonic sensors circuit diagram

Testing was conducted with various obstacle placements to ensure accurate readings. The ultrasonic sensors consistently detected obstacles up to 400 cm away, with an accuracy margin of ±1 cm. Two of the sensors were used for a wider range of obstacle detection. The testing process is demonstrated in in Figure 6-2 where one of our team members placed an arm about 25 cm (measured using a measuring tape) away from the sensors. Figure 6-3 demonstrates successful detection of obstacle approximately 25cm from the robot from both sensors.

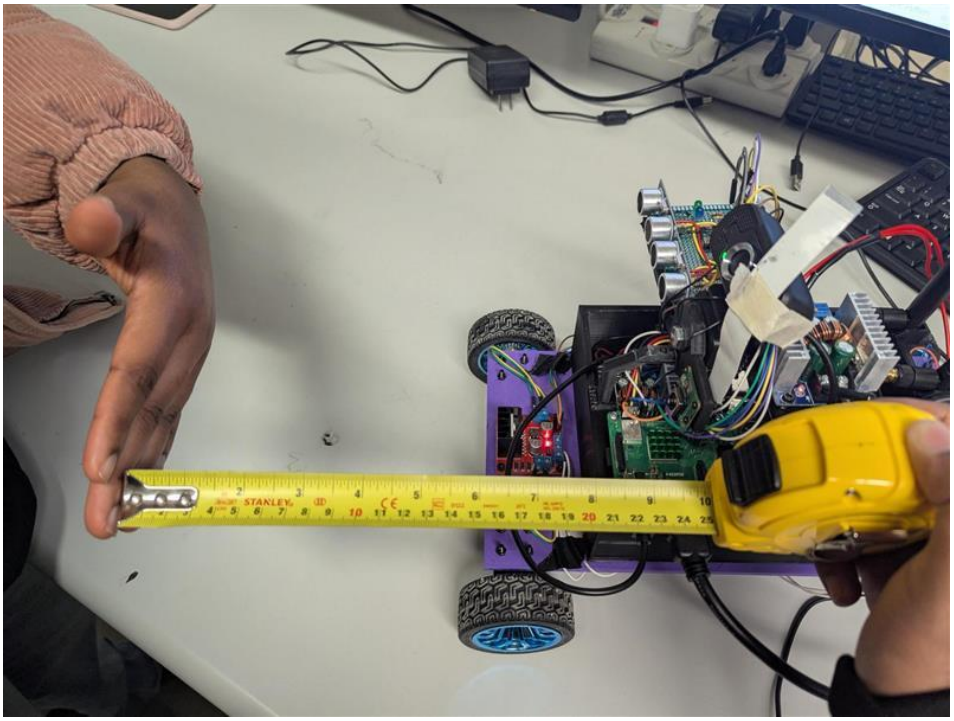


Figure 6-2: Testing the ultrasonic sensor 25 cm away from a measuring tape.

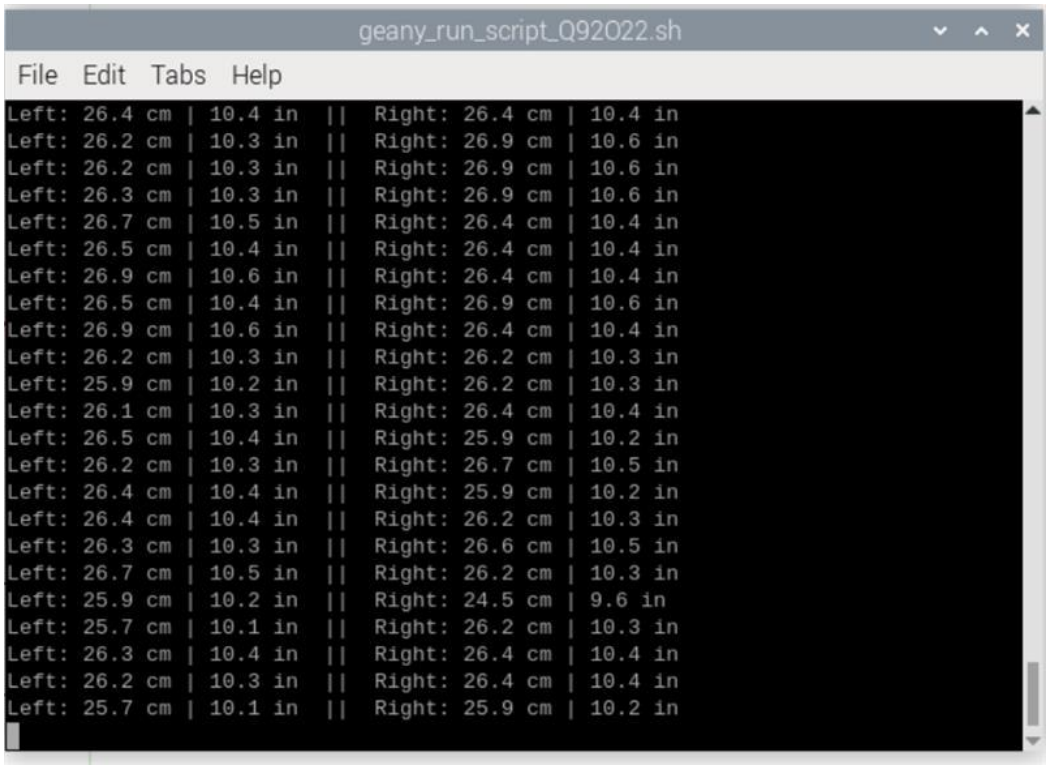


Figure 6-3: Both ultrasonic sensors working and outputting obstacle distance successfully

6.1 Concurrent Data Processing with Computer Vision

As ServoSentry collects both ultrasonic and vision data, it processes these inputs concurrently to enhance perception. The ultrasonic sensors detect obstacles for safe navigation while the Jetson Nano processes vision data for object detection. The combination of these inputs enhances the environmental awareness of ServoSentry.

The camera from the vision system is connected to the Jetson Nano via a USB cable and captures real time images for object detection and tracking. The Jetson Nano processes these images using the YOLOv3 model, identifying and localizing objects of interest. Once an object is detected, the bounding box coordinates are transmitted to the Pi over UART, enabling the robot to make navigation decisions based on both vision and ultrasonic data. Figure 6-4 below shows a flowchart that describes how the UART and ultrasonic data are processed on the Pi.

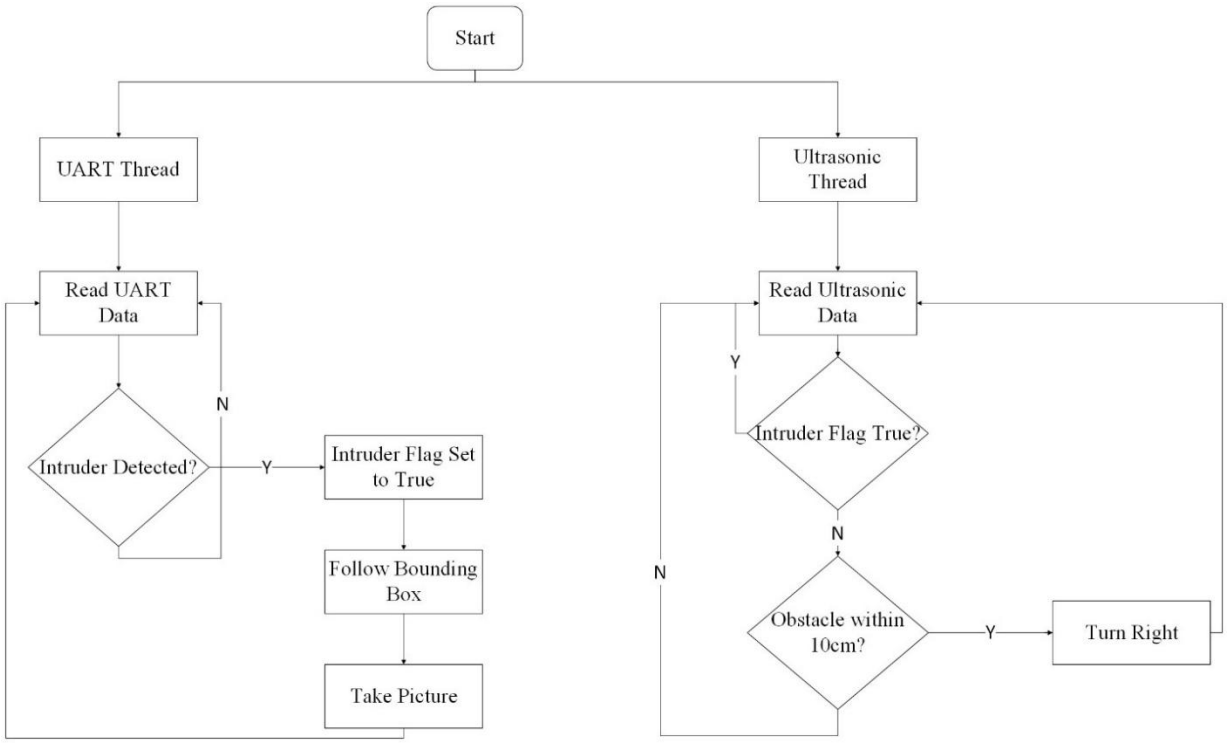


Figure 6-4: Flowchart showing Pi processing of Vision and Ultrasonic data.

A thread for the image received over UART is created at the start along with a thread for the ultrasonic sensors' distance measurement. Both threads run concurrently, ensuring that visual and distance data are processed in real time. The image processing thread decodes the bounding box coordinates received over UART from the Jetson Nano and determines the location of a detected intruder. At the same time, the ultrasonic thread continuously measures distances to nearby obstacles and alerts the robot when the object is within 10cm. However, an intruder flag is used to allow the UART thread have priority over the ultrasonic thread to ensure the robot prioritizes processing visual data for tracking intruders. With the sensor processing in place, the next aspect to explore is the electrical system, which powers and interfaces with these components to bring the robot's functionality to life.

7 Electrical System

The electrical system designed provides power to drive the core systems and functionalities of ServoSentry. This section outlines a simple configuration of the power distribution connection from the 11.1V LiPo battery to the DC-DC converters and finally to the components used to build and develop the computer vision, control and locomotion systems of ServoSentry.

To provide power to the systems, a 11.1V LiPo battery is connected to two buck DC-DC converters. The buck converters step down the input 11.1V to 5V and 6V, the 5V is supplied to the Nvidia Jetson Nano and the Raspberry Pi and 6V is supplied to the respective motor drivers that control the wheels and legs movements. A block diagram which conveys the connection between the system is shown in Figure 7-1.

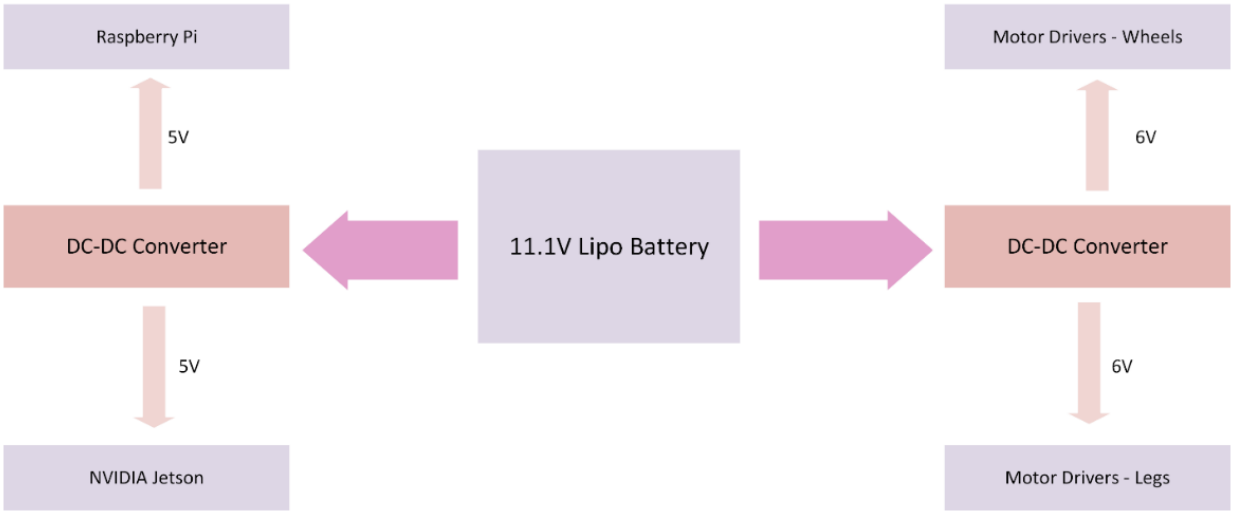


Figure 7-1 Power Distribution

Table 7-1 highlights the details of the voltage and current specifications for each system. These specifications are used to provide an estimate of how long the 11.1V LiPo battery can supply power to ServoSentry until it needs to be recharged.

Table 7-1: Voltage and current specifications for key components

System	**Component**	**Voltage and Current Specs.**
Computer Vision	Jetson Nano	5V at 4A
Control Hub	Raspberry Pi	5V at 2.5A
Wheel Locomotion	Motor Drivers	6V at 400mA
Quadrupedal Locomotion	Motor Driver	6V at 0.7A while standing 6V at 1.64A while walking

The total current of the 5V system is 6.5A, resulting in 32.5W. The 6V wheel motors use 2.4W and the legs use 4.2W while standing and 9.84W for walking. For the quadrupedal locomotion, the power consumption totals to 36.7W while ServoSentry is stationary and 42.34W when it is in motion. With the 11.1V 2200mAh battery, the available energy is 24.42Wh. Dividing this by the total system power gives an estimated runtime of around 0.6 hours (about 35 minutes) while walking, or about 0.7 hours – 40 minutes, when standing. The power consumption for the wheel locomotion is 34.9W, this translates to an estimated runtime of about 0.7hours - 40 minutes.

7.1 Interface board

The interface board shown in Figure 7-2 serves three purposes: to handle the ultrasonic sensors, to mount the switch circuitry for swapping locomotion systems, to provide connections points for both locomotion systems' signal wires. The row of pin headers at the bottom of the board, in order from left to right are power connections for the interface board, connections to the ultrasonic sensors, and connections to the switch circuit. All these pins connect to the Raspberry Pi. There is also a blue and green LED that indicates whether the quadrupedal or wheeled system is active.

The pin headers in the centre of the board provide the locomotion system connection points, as the left grouping of pin headers connects to the Raspberry Pi, while the right grouping of pins connects to the locomotion systems. This setup was done rather than directly connect the locomotion signals to the Raspberry Pi because we could not get a continuous row of 4 or 5 pins to make an easy connection; instead, the Raspberry Pi's pins are spread out and unlabelled. Additionally, because the pins are unlabelled it takes some time and is error prone. Thus, using

the interface board facilitates rapid rewiring between the locomotion systems as required by our modular design.

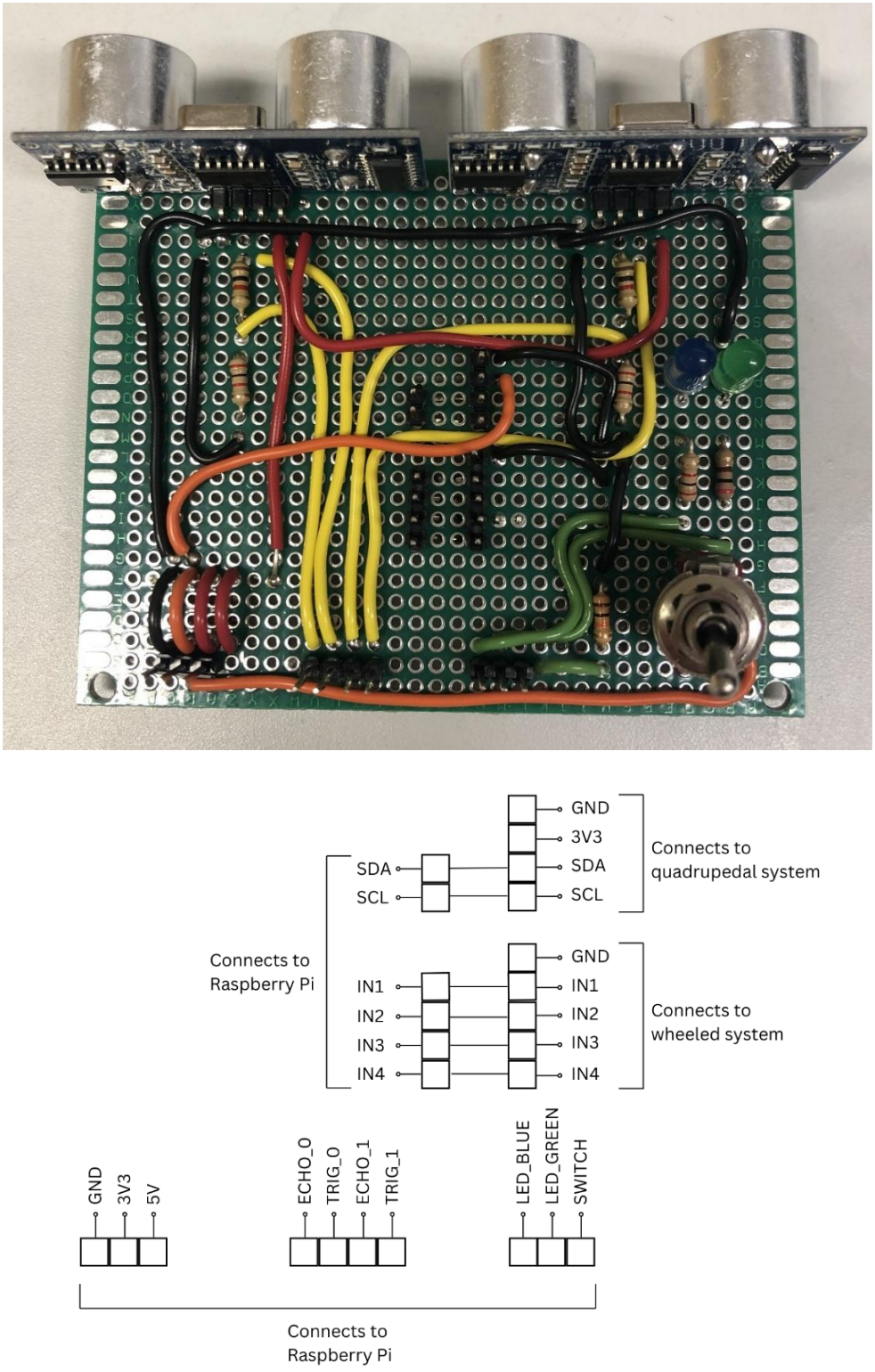


Figure 7-2 Interface board and its pinout to the Raspberry Pi and locomotion systems.

8 Software Implementation

8.1 Main control code description

The main control code that defines ServoSentry is run on the control hub as implemented by the Raspberry Pi. The aim of the control code is to integrate various components of ServoSentry such as motor control, ultrasonic sensors, a camera and a toggle switch to enable the robot detect intruders, avoid obstacles, and switch between wheel and leg movement modes.

The robot utilizes UART to receive data about detected objects. A dedicated function parses the incoming data and extracts the object class and bounding box coordinates received from the vision system. This data is then used to determine the position and size of the detected object which is important for controlling the robot's movement. In the dedicated UART receive function, a picture is taken by the robot when the object class received is "intruder" and the robot then adjusts its movement to continue following the object.

Another function which further decodes the bounding box coordinates received by the robot is used to ensure the detected object is at the centre of the robot's camera frame. The centre of the bounding box is obtained via the formula $c = x + \frac{w}{2}$, where x and w are the x-coordinate and width of the bounding box respectively. Also, the area of the bounding box is calculated using the formula $A = width \times height$. These two derived variables allow the robot to align itself with the detected object if the object is off-centre or if the object is too close.

For the ultrasonic sensors, a function is used to process the ultrasonic data received. The distance between the robot and an obstacle is calculated via the formula $D(cm) = 0.5 * (Travel\ Time\ (s) * 34300\ (cm/s))$, where the Travel Time is the time taken for a trigger signal to be sent and an echo to be returned [43]. The travel time is multiplied by the speed of sound and then the product is halved since the travel is a two-way trip. If any obstacle is within a 10cm range, the robot turns to avoid it. However, when an intruder is seen by robot, following the intruder takes priority.

As mentioned in Section 6.1, the functions responsible for the UART data processing and ultrasonic data processing are run in separate threads to allow for parallel processing of data. Another thread which is responsible for switching the robot between its two locomotion modes is also ran separately which brings ServoSentry's total thread count to three. The mode switching

thread monitors the state of a toggle switch and updates a global variable which executes the desired locomotion control for the robot.

8.2 Decision-Making Algorithm

At the core of the system, the Jetson Nano serves as the primary vision processing unit, running YOLOv3 for object and intruder detection. The detected bounding box coordinates are then transmitted via UART to the Raspberry Pi, which processes these coordinates and determines the robot's movement strategy. The Raspberry Pi also receives real-time data from the ultrasonic sensors, allowing it to integrate both vision-based detections and physical obstacle avoidance.

To achieve seamless operation, the Jetson Nano is configured to boot up and automatically initialize the object detection script. This eliminates the need for manual intervention and ensures that the detection pipeline is operational as soon as power is supplied. The locomotion system, whether wheeled or quadrupedal, is controlled based on a decision-making algorithm that prioritizes intruder tracking while maintaining safe navigation.

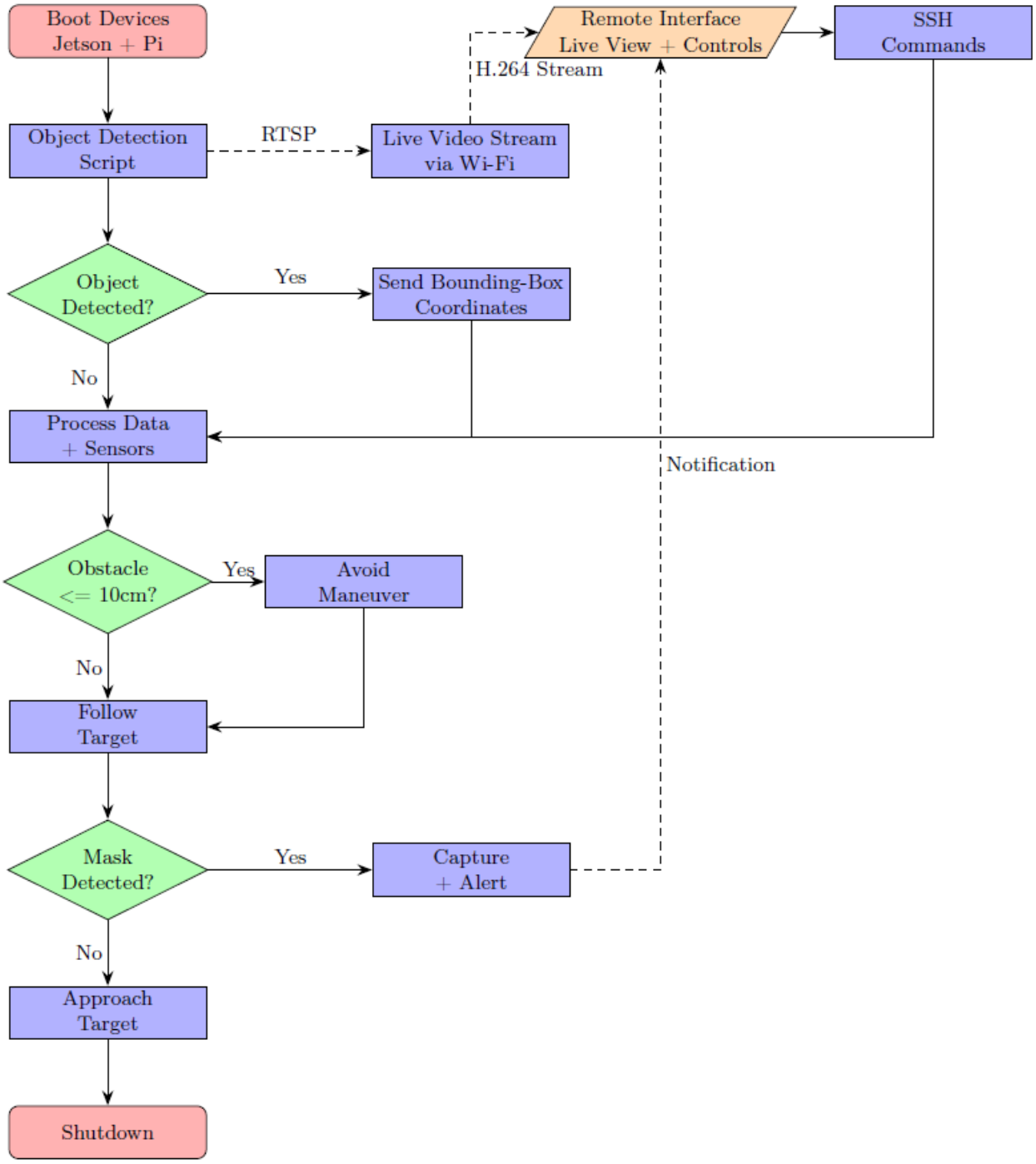


Figure 8-1: High-Level System Diagram of ServoSentry, illustrating the data flow between the Jetson Nano, Raspberry Pi, ultrasonic sensors, and locomotion subsystems.

Figure 8-1 shows the algorithm ServoSentry uses to navigate and track intruders is designed to ensure autonomous operation. Irrespective of the locomotion system in play, the algorithm compares the calculated bounding box centre to the centre of camera frame and if the bounding box's centre is less than that of the frame's centre, the robot turns left and vice versa.

In addition, if the intruder is centred on the robot's frame, the robot moves forward to approach it depending on how close it is to the intruder. A threshold set on the area of the bounding box ensures that robot does not come too close to the intruder, thereby allowing for a picture of the intruder to be taken.

ServoSentry is designed to prioritize obstacle avoidance only when it is not actively tracking an intruder. When an obstacle is detected, the robot turns right and continues to do so until the obstacle is no longer in its path. However, if the robot is following an intruder and encounters an obstacle, it follows a different set of rules:

- If the intruder is still detectable (i.e., the bounding box coordinates are still being received via UART), the robot prioritizes tracking the intruder over avoiding the obstacle. It continues to adjust its movement based on the intruder's position, even if an obstacle is present.
- If the intruder is no longer detectable (e.g., the bounding box data is lost), the robot immediately switches to obstacle avoidance mode. It turns right and continues to do so until the obstacle is cleared.

This ensures that the robot always attempts to fulfill its primary objective to track intruders.

9 Integrated System Test and Results

9.1 Obstacle Avoidance Tests

The obstacle avoidance system performed reliably in detecting and responding to solid objects, successfully navigating around obstacles using ultrasonic sensors. Single and dual-sensor tests showed accurate detection and effective path adjustments. The system also correctly prioritized intruder detection, integrating vision-based tracking with ultrasonic distance verification to ensure safe stopping. While netted boundaries posed challenges due to ultrasonic pulses passing through gaps, this scenario is uncommon in most real-world applications, and alternative sensors like LiDAR could be explored for enhanced detection. Table 9-1 shows the successful detection of tests OB-1 and OB-2.

Table 9-1: Obstacle Avoidance Test Results

Test ID	**Test Scenario**	**Expected Outcome**	**Actual Outcome**	**Notes**
OB-1	Single Ultrasonic Sensor Object placed 15 cm in front of sensor	Robot should detect the object at 15 cm and either stop or move away from the obstacle	Robot detected the object consistently at 15 cm, then turned right to avoid collision	Voltage divider ensured 3.3V logic level for Echo pin; sensor accuracy was within ±1 cm
OB-2	Dual Ultrasonic Sensors Object placed 20 cm in front of each sensor at different angles	Robot should detect objects on both sensors and choose a clear path based on the sensor returning the smaller distance	Robot detected both obstacles and moved in the direction of the greater available space	Demonstrated effective coverage of a wider field of view; minimal overlap in sensor blind spots
OB-3	Netted Boundary Robot runs in enclosed drone testing area	Robot should detect netted boundary and turn away upon detection	Robot failed to detect the netted boundary; ultrasonic pulses passed through net openings	Net-like surfaces pose a challenge for ultrasonic reflection. Alternative sensors (e.g., LiDAR or IR) may be required to detect thin or permeable materials
OB-4	Intruder Scenario Robot moves forward, intruder steps in front	Robot should detect intruder via mask detection, ultrasonic sensor should confirm presence at 20 cm, then stop or turn	Intruder detection triggered the system to prioritize bounding box data; ultrasonic sensor recognized close range, prompting the robot to halt	Vision system's bounding box data had priority; ultrasonic sensor provided distance verification, ensuring the robot stopped before collision

9.2 Intruder Detection and Tracking

To ensure reliability, we conducted extensive intruder detection testing on these three fronts:

- Model Accuracy: The mask detection system was tested on various individuals with different facial coverings to verify classification precision.
- Real-time Processing: Bounding box filtering mechanisms were applied to maintain smooth tracking and avoid latency spikes.
- Data Transmission: Debugging logs confirmed that only intruder-related information is sent when a masked person is detected, minimizing unnecessary data exchange.

Intruder detection and tracking with the wheel locomotion system and the quadrupedal locomotion system are discussed next.

9.2.1 Wheel System

Testing the intruder tracking capabilities with the wheel system revealed that the robot was able to successfully follow a masked individual. The system quickly generates bounding box data that is sent over UART to the control hub, allowing the wheeled platform to adjust its course in real time and keep up with the target. In several test runs, the robot consistently kept the intruder centered in its field of view and adjusted its path as the target moved.

The robot also demonstrated reliable maneuvering in all directions while avoiding obstacles effectively, thanks to the ultrasonic sensor set at a 20 cm threshold. Discrete movement control helped reduce overshoot and improved the system's responsiveness, even though there were some minor delays caused by transmission speed limitations. Communication over UART for data transmission was steady, confirming that the integration of the subsystems is solid.

Despite occasional delays, likely due to processing latency in the camera module or minor communication delays, the robot maintained its tracking objective throughout the tests. This consistent performance highlights the potential of the system for real-world applications. The fully integrated wheel system is shown in Figure 9-1.

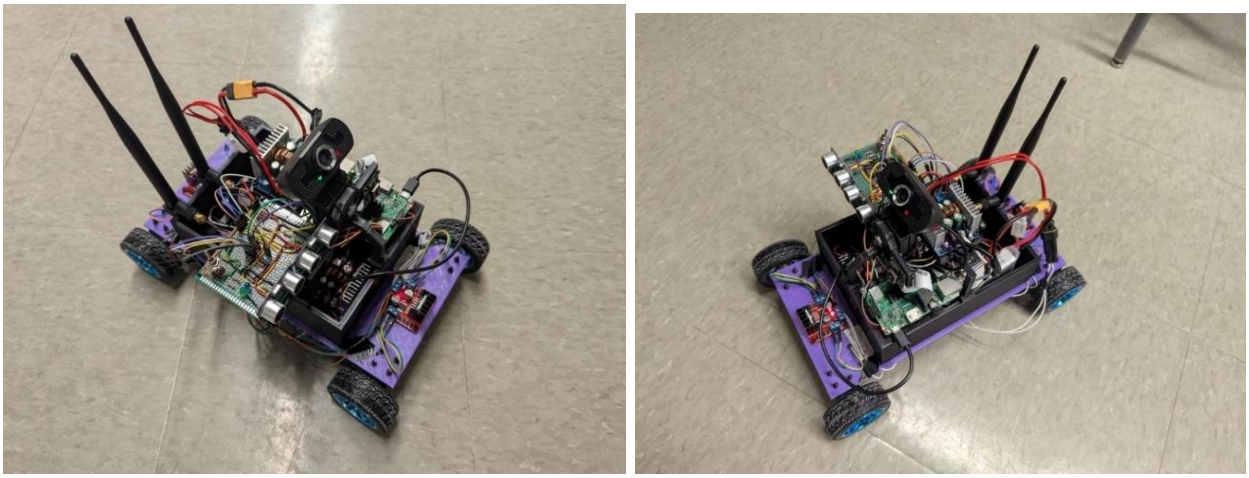


Figure 9-1: Final Integrated Wheel System

9.2.2 Quadrupedal System

Like the wheel system, the quadrupedal system demonstrated effective intruder tracking, successfully following a masked target in real time. Using continuous movement control, tracking was improved although occasional delays occurred due to processing and communication constraints.

Despite these minor interruptions, the system maintained its tracking functionality throughout testing, reinforcing the effectiveness of the integrated components. The fully integrated quadrupedal system is shown in Figure 9-2.

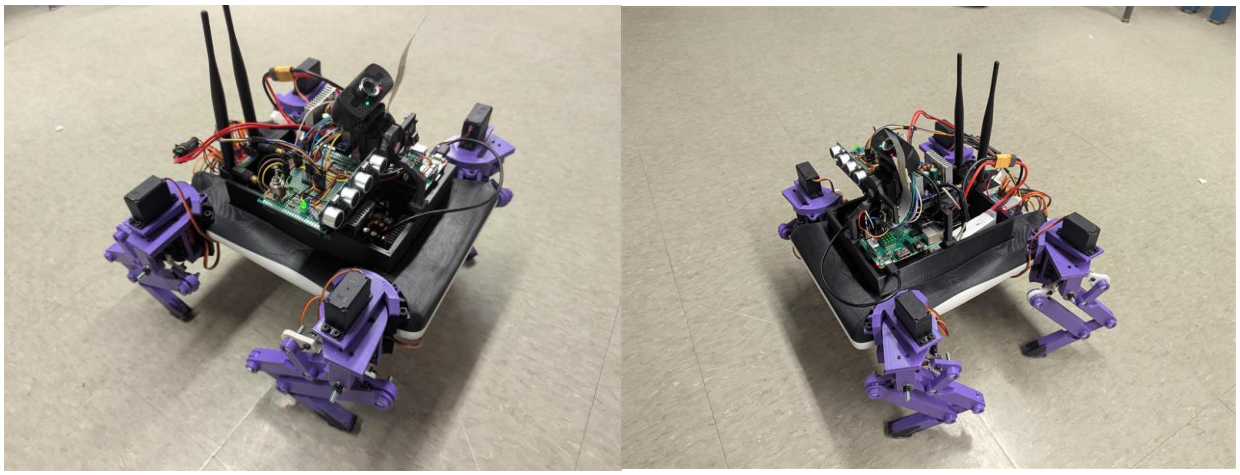


Figure 9-2: Final Integrated Quadrupedal System

9.3 Performance Metrics

ServoSentry successfully achieved its core objectives across multiple subsystems. The computer vision module surpassed the required 4 FPS and operated at a respectable 14.8 FPS, with intruder detection accuracy reaching 70%. The detection accuracy was validated through multiple test scenarios where masked and unmasked individuals were introduced in varying lighting conditions to ensure reliable classification. Communication between the NVIDIA Jetson Nano and Raspberry Pi was robust, ensuring that all bounding box and classification data was reliably transmitted over UART. The data transmission was verified by comparing the sent and received coordinates and object classifications over multiple test runs. The ultrasonic sensor also exceeded expectations, accurately measuring distances up to 500 cm compared to the 100 cm minimum requirement. This was tested by placing objects at incremental distances and measuring the sensor's response accuracy.

The wheeled system performed admirably, reaching a speed of 0.52 m/s and successfully executing forward, backward, and turning maneuvers. The speed test was performed by getting the robot to move forward for 0.5s and then measuring the distance travelled which turned out to be 0.26m. Integrated testing of the wheeled configuration revealed effective obstacle avoidance (with a 20 cm sensor threshold) and accurate intruder tracking. During these tests, discrete motion commands implemented to mitigate overshoot from continuous motion proved essential in maintaining effective tracking.

Testing of the quadrupedal system confirmed that it could safely support the robot's load, even though its movement speed (1.63 cm/s) was lower than anticipated as explained in Section 3.2.8. Importantly, while the battery endurance in continuous motion tests was observed to be 35–40 minutes, when the main control code operated in a discrete movement mode triggered by intruder detection, if both the wheeled and leg systems easily sustained operation for up to 2 hours.

Overall, ServoSentry demonstrated strong autonomous navigation, tracking, and adaptability in both configurations. These results not only validate the integrated performance of the system but also lay a solid foundation for future refinements in efficiency, modularity, and sensor integration.

Table 9-2: Table of Performance Metrics

Module	Task	Minimum Requirements	Measurement Unit	Metrics	Outcomes (pass/fail)
Computer Vision	Object Detection Speed	Detection of one frame within 0.25s (4 FPS)	Time (s)	Frames Per Second (FPS)	0.067 seconds per frame at 14.8 FPS
	Detection Accuracy	≥ 70% detection rate for intruders	Percentage (%)	Accuracy rate	70% for intruders, 70% for other people and 55% for other objects (Pass)
Software	Object Recognition Classes	Detect at least 11 object classes across two datasets	Number of classes	≥ 11 classes	10 classes using the COCO dataset and 2 classes using the masked-model dataset (total = 12 classes) (Pass)
	Data transmission between vision and control system	Correctly send bounding box coordinates and object class data over UART	Data format integrity (e.g., object label and (X, Y, h, w) coordinates)	All data being transmitted is received by raspberry pi.	The raspberry pi receives all data being sent -> Object class detected & (X, Y, h, w) coordinates (Pass)
Sensor	Ultrasonic Range	The ultrasonic sensor should reliably measure distances of at least 100 cm	Distance (cm)	≥ 100cm	Sensor successfully measures up to 500 cm (Pass)
Communication	Wi-Fi Connectivity for Remote access	Maintain reliable communication over a distance of at least 1 m	Distance (m)	≥ 1m	Wifi module installed, and remote communication established within 1m. (Pass)
Power Supply	Battery Endurance	The battery should power the robot for at least 2 hours	Time (hours)	≥ 2 hours	Quadrupedal system: 35 minutes when in motion. (Fail) 40 minutes when stationary. (Fail) Wheeled System: 40 minutes when in continuous motion. (Fail)
Locomotion	Quadrupedal and wheeled load capacity	Must support the mass of robots' locomotion.	Support robot's total mass (excluding motors)	>1.75kg	Quadrupedal and wheeled system more than 1.75kg successfully supported (pass)
	Quadrupedal and wheeled movement.	Robots can move and turn without falling over.	Speed (m/s)	>0.25 m/s	Quadrupedal system: 1.63 cm/s (Fail) Wheeled system: 0.52 m/s (Pass)
Integration	Swap carrier board between leg and wheels system	The carrier board must be swappable within 15 seconds	Time (s)	<15 s	24.63s (Fail)
	Autonomous Navigation and System Integration	Integrated system must reliably execute autonomous navigation, obstacle avoidance, and intruder tracking with minimal response delays.	Response delay	<1s	Quadrupedal system: reliable autonomous navigation, obstacle avoidance, and intruder tracking. <1s delay (Pass) Wheeled system: reliable autonomous navigation, obstacle avoidance, and intruder tracking. <1s delay (Pass)

10 Future Enhancements

An abundance of knowledge was gathered over the course of designing ServoSentry, all of which we are eager to apply to future endeavours in robotics, computer-vision, and sensor integration. We now discuss enhancements that can be made to ServoSentry for improving performance, and additional functionality that would be worth exploring.

10.1 Wheeled System

The wheeled system's hardware has been the biggest improvement to allow for more all-terrain applications like patrolling quarries or construction sites. The rigidity of the wheel's connection to the chassis does not allow for much flexibility which would cause the computer vision system to vibrate when going over rough terrain and decrease the detection accuracy. Furthermore, obstacles like small rocks or stairs would restrain ServoSentry's movements. This could be solved by exploring a more all-terrain design like having a proper suspension system or having a more flexible six wheeled design.

For patrolling, it would have been best to implement a proper pathing procedure to allow ServoSentry to patrol along a predetermined route. This would have allowed for a wider range of coverage for intruder detection such as covering an entire construction site rather than a small corner. This could have been accomplished by implementing a 3D LiDAR sensor to detect and avoid obstacles and having ServoSentry identify where it is with respect to a base station to properly patrol a specific route.

10.2 Quadrupedal System

Beginning with overall design workflow, certain quality of life items should've been completed much earlier in the project like soldering all the wires properly and testing the system directly with a Raspberry Pi rather than an Arduino, as was initially done. Future work should also focus on using more advanced testing apparatus, like building a rig to test a leg design and measuring properties like traction, output power, and to test control algorithms in finer detail.

For hardware, it would be interesting to switch from servo motors to DC motors, as they provide a higher degree of control and are typically higher quality than servo motors. The trade-off is that more mechanical and electrical systems are needed to increase the motor torque and attain position control. Incorporating sensors like pressure sensors on the feet, and a gyroscope

would help the robot move more accurately and provide information about its balance. A study of inverse kinematics should be undertaken to leverage more advanced techniques so that more complicated mechanisms can be modelled and controlled.

10.3 Vision System

One limitation of our vision system is that the masked detection module treats every masked face as an intruder. In a cold city like Winnipeg, where mask usage is common, this leads to false alerts. To address this, we have developed a face recognition module that differentiates between individuals based on unique facial features.

The system leverages the facenet-pytorch framework, using MTCNN for precise face detection and InceptionResnetV1 for extracting detailed facial embeddings. A Support Vector Classifier (SVC) is trained on a dataset containing positive samples (authorized users) and negative samples (unknown individuals) to improve accuracy in identifying known personnel while flagging unrecognized faces as potential intruders [30], [41]. It only has not been implemented to the Jetson Nano based processor we have due to the limitations discussed.

This enhancement broadens ServoSentry's application as a multi-purpose robot. For instance, in a museum surveillance scenario, the system could be configured with a pre-approved list containing only security personnel. Any unrecognized face would trigger an alert and initiate a security response.

While the face recognition module has demonstrated high accuracy on a powerful computing system, real-time deployment on the Jetson Nano remains a challenge. Future optimization efforts include:

- Implementing TensorRT for faster inference speed.
- Using optimized NumPy and OpenCV builds to reduce computational overhead.
- Reducing the feature extraction model size for improved efficiency.

Additionally, upgrading from the 2018 OKdo Jetson Nano to a more recent NVIDIA Jetson Nano with better support for modern Python packages may enhance compatibility and performance. These improvements will help bridge the gap between desktop-based testing and real-time deployment, ensuring ServoSentry can reliably execute face recognition in practical

applications. The latest implementation is available in our GitHub repository [44], where further refinements are ongoing.

Overall, future work on ServoSentry will focus on refining its adaptability, efficiency, and real-time processing capabilities. Key areas of improvement include enhancing the locomotion systems to better navigate complex terrains, optimizing the face recognition module for deployment on the Jetson Nano, and integrating additional sensors such as LiDAR for improved spatial awareness. Further software optimization efforts will involve implementing TensorRT acceleration, reducing model complexity, and refining object tracking algorithms to ensure more reliable real-time performance. Additionally, to ensure ease of access and continued development, Appendix B. Code Listings and Test Data, provides links to all relevant GitHub repositories within our organization, detailing the latest code implementations and future updates to ServoSentry. These enhancements aim to bridge the gap between prototype and practical deployment, ensuring that ServoSentry remains a robust and versatile autonomous system.

11 Conclusion

In summary, the project successfully achieved a robust and modular design by integrating both wheeled and quadrupedal locomotion systems. These systems are controlled by a Raspberry Pi and meets nearly all our performance metrics. The wheeled system not only supports the required load of at least 1.75 kg but also achieves the necessary speed targets, having been refined through several iterations of the chassis design. Similarly, the quadrupedal system supports the required 1.75 kg load and can walk and turn. The wheeled system met our speed requirement of 25 cm/s, as it achieved 52 cm/s while the quadrupedal system did not meet that requirement, only moving at 1.63 cm/s. The carrier board, securely attached using eight magnets, facilitates easy interchange between locomotion modes and includes a dedicated mount for additional electronics, ensuring flexibility and ease of maintenance.

On the software side, the computer vision module accomplished its objectives by effectively detecting masked individuals and performing facial recognition. Running at speeds between 10 and 15 FPS with OpenCV's CUDA acceleration and PyTorch, it provides real-time data crucial for autonomous navigation and intruder tracking. Together, these achievements highlight a balanced integration of advanced computer vision, versatile locomotion, and modular hardware, establishing a strong foundation for further enhancements and practical real-world applications.

Appendices

Appendix-A

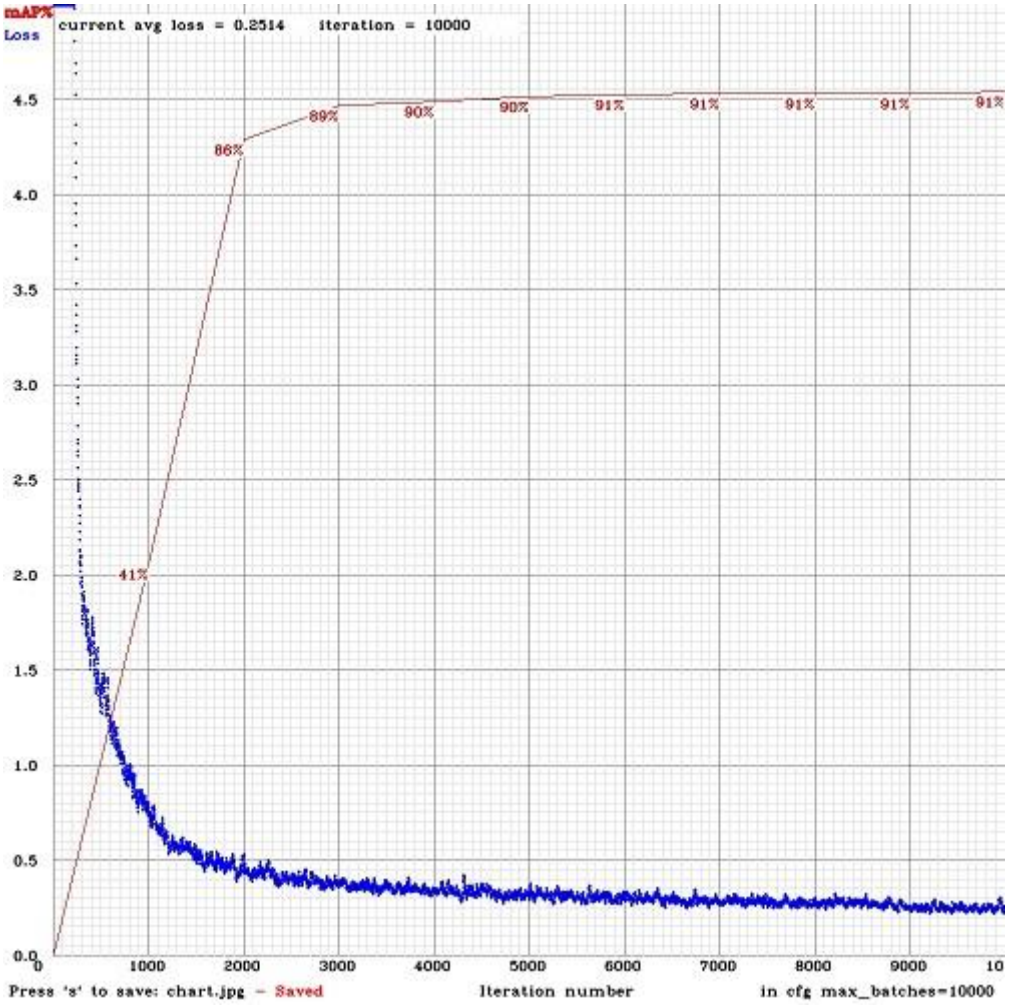


Figure 0-1: YOLOv3 Loss and mAP curves from Darknet's training process. The decreasing loss and increasing mAP indicate successful training and improved object detection performance [37]

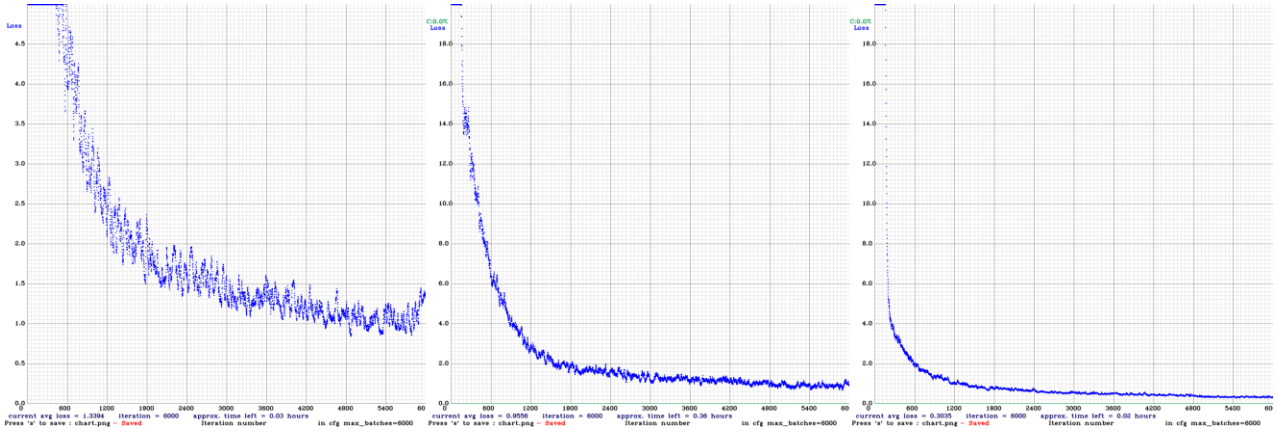


Figure 0-2: Loss and mAP curves for the mask detection model, indicating successful convergence over 6000 iterations [36]

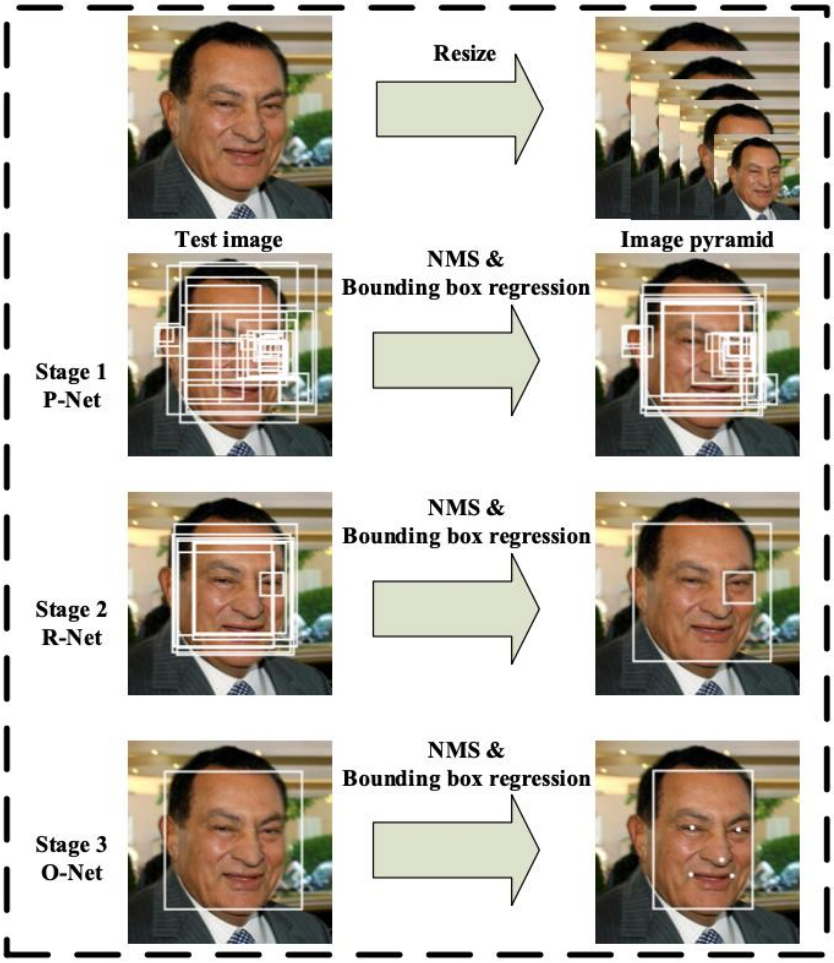


Figure 0-3: The Pipeline of the MTCNN Framework that Includes Three-stage Multi-task DNNs [40].

Appendix B. Code Listings and Test Data

A GitHub organization was created for ServoSentry to house all software used in the project. The following git repositories are included:

- UM-Capstone-G08-2025/legControl:
  Python package for controlling the quadrupedal system of the capstone project.
- UM-Capstone-G08-2025/Computer-Vision:
  Detects objects in the webcam, adds visual labels to the video stream, and sends the stream to client devices.
- UM-Capstone-G08-2025/Face-Recognition-Jetson
  This is a face recognition workspace to be implemented on Jetson Nano.
- UM-Capstone-G08-2025/RaspberryPi_Control
  Controls the directional movements of ServoSentry.
- UM-Capstone-G08-2025/Server
  Proxy server to connect the robot and client devices.

Reference List

[1] "Jetson Nano Developer Kit," [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed 7 March 2025].

[2] "Edge computing," [Online]. Available: https://en.wikipedia.org/wiki/Edge_computing. [Accessed 7 March 2025].

[3] "NVIDIA, “CUDA Zone,”," [Online]. Available: https://developer.nvidia.com/cuda-zone. [Accessed 7 March 2025].

[4] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *arXiv preprint*, Ithaca, 2016.

[5] NVIDIA, "Jetson Nano Developer Kit User Guide," 2023. [Online]. Available: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/NV_Jetson_Nano_Developer_Kit_User_Guide.pdf. [Accessed 18 February 2025].

[6] "NVIDIA, “JetPack SDK,”," [Online]. Available: https://developer.nvidia.com/embedded/jetpackNVIDIA, “JetPack SDK,”. [Accessed 7 March 2025].

[7] "NVIDIA, “cuDNN,”," [Online]. Available: https://developer.nvidia.com/cudnn. [Accessed 7 March 2025].

[8] "NVIDIA, “TensorRT,”," [Online]. Available: https://developer.nvidia.com/tensorrt. [Accessed 7 March 2025].

[9] "OpenCV, “OpenCV,”," [Online]. Available: https://opencv.org/. [Accessed 7 March 2025].

[10] FRIWOL, "FRIWOL USB HD1080P 2MP Webcam PC Camera with Absorption Microphone MIC for SKYPE Android TV ROTATABLE Computer Camera 1080P HD 2K(WM12), Black (Brixwm07-webcam)," [Online]. Available:

https://www.amazon.in/FRIWOL-Absorption-Microphone-ROTATABLE-Computer/dp/B08DMJYGMN. [Accessed 21 February 2025].

[11] "PiShop.ca," Raspberry Pi 3 - Model B+, [Online]. Available: https://www.pishop.ca/product/raspberry-pi-3-model-b-plus/. [Accessed 19 February 2025].

[12] "Raspberry Pi 3 Model B+," [Online]. Available: https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf. [Accessed 19 February 2025].

[13] H. Tech, "HC-SR04 Ultrasonic Sensor Datasheet," 2023. [Online]. Available: https://www.handsontec.com/dataspecs/HC-SR04-Ultrasonic.pdf. [Accessed 18 February 2025].

[14] N. Developer, "JetPack Linux," [Online]. Available: https://developer.nvidia.com/embedded/jetson-linux. [Accessed 14 March 2025].

[15] kernel.org, "The Linux Kernel Archives," [Online]. Available: https://kernel.org. [Accessed 14 March 2025].

[16] releases.ubuntu.com, "Ubuntu 18.04.5 LTS (Bionic Beaver)," [Online]. Available: https://releases.ubuntu.com/18.04/. [Accessed 14 March 2025].

[17] N. Developer, "CUDA Toolkit," [Online]. Available: https://developer.nvidia.com/cuda-toolkit. [Accessed 14 March 2025].

[18] Python.org, "Python Release Python 3.6.0," [Online]. Available: https://www.python.org/downloads/release/python-360/. [Accessed 14 March 2025].

[19] Numpy, "numpy," [Online]. Available: https://github.com/numpy/numpy. [Accessed 14 March 2025].

[20] openCV, "opencv-python," [Online]. Available: https://github.com/opencv/opencv-python. [Accessed 14 March 2025].

[21] matplotlib, "matplotlib," [Online]. Available: https://github.com/matplotlib/matplotlib.. [Accessed 14 March 2025].

[22] pyserial, "pyserial," [Online]. Available: https://github.com/pyserial/pyserial. [Accessed 14 March 2025].

[23] PyTorch, "pytorch," [Online]. Available: https://github.com/pytorch/pytorch. [Accessed 14 March 2025].

[24] PyTorch, "torchvision," [Online]. Available: https://github.com/pytorch/vision. [Accessed 14 March 2025].

[25] scikit-learn, "scikit-learn," [Online]. Available: https://github.com/scikit-learn/scikit-learn. [Accessed 14 March 2025].

[26] python-pillow, "Pillow," [Online]. Available: https://github.com/python-pillow/Pillow. [Accessed 14 March 2025].

[27] joblib, "joblib," [Online]. Available: https://github.com/joblib/joblib. [Accessed 14 March 2025].

[28] opencv, "opencv," [Online]. Available: https://github.com/opencv/opencv. [Accessed 14 March 2025].

[29] J. Redmon, "darknet," [Online]. Available: https://github.com/pjreddie/darknet. [Accessed 14 March 2025].

[30] Timesler, "facenet-pytorch," GitHub repository, [Online]. Available: https://github.com/timesler/facenet-pytorch. [Accessed 18 February 2025].

[31] J. Bruton, "openDog V3: How I Made it Walk," 14 December 2021. [Online]. Available: https://www.youtube.com/watch?v=eKZIJwJBjEs&list=PLpwJoq86vov8uTgd8_WNgBHFpDYemO-OJ&index=5. [Accessed 20 February 2025].

[32] Eddie, "Quadruped Robot Dog-MATLAB Simulate Inverse Kinematic Trot Step," 28 February 2023. [Online]. Available: https://www.youtube.com/watch?v=KwLMVeGbPdg. [Accessed 20 February 2025].

[33] asd_lab, "You Won't Believe What This DIY Leg Mechanism is Capable of! Quadruped Robot Building," 26 October 2022. [Online]. Available: https://www.youtube.com/watch?v=8a_HKH8vGjc&list=LL&index=6&t=116s. [Accessed 20 February 2025].

[34] D. Schilling, "How Can I Find the Points at Which Two Circles Intersect?," StackExchange, 16 November 2018. [Online]. Available: https://math.stackexchange.com/questions/256100/how-can-i-find-the-points-at-which-two-circles-intersect. [Accessed 24 February 2025].

[35] "YOLO Algorithm for Object Detection Explained [+Examples]," [Online]. Available: https://www.citationmachine.net/ieee/cite-a-website/confirm. [Accessed 19 February 2025].

[36] Cansik, "yolo-mask-detection," [Online]. Available: https://github.com/cansik/yolo-mask-detection. [Accessed February 2025].

[37] AlexeyAB, "darknet," [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed February 2025].

[38] "A Deep Dive Into Non-Maximum Suppression (NMS)," Built In, [Online]. Available: https://builtin.com/machine-learning/non-maximum-suppression. [Accessed 19 February 2025].

[39] "Unlocking the Power of Facial Blurring in Media: A Comprehensive Exploration and Model Comparison," Medium, 18 September 2023. [Online]. Available: https://medium.com/towards-data-science/unlocking-the-power-of-facial-blurring-in-media-a-comprehensive-exploration-and-model-comparison-261031603513. [Accessed 21 February 2025].

[40] K. Zhang, Z. Zhang, Z. L. S. M. IEEE and a. Y. Q. S. M. IEEE, "Joint Face Detection and Facial Expression Recognition with MTCNN | IEEE Conference Publication | IEEE Xplore," IEEE, [Online]. Available: https://ieeexplore.ieee.org/document/8110322/. [Accessed 21 February 2025].

[41] Scikit-learn, "Support Vector Machines," Scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/svm.html. [Accessed 21 February 2025].

[42] NVIDIA, "Jetson Nano Developer Kit User Guide," NVIDIA Developer, [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit. [Accessed 21 February 2025].

[43] L. Pfahler, "YouTube. Connecting an HC-SR04 Ultrasonic Sensor to a Raspberry Pi," 12 January 2023. [Online]. Available: https://www.youtube.com/watch?v=C02oB1n7rcg&t=1s. [Accessed 21 February 2025].

[44] UM-Capstone-G08-2025, "Computer Vision for ServoSentry, Github Repository," [Online]. Available: https://github.com/UM-Capstone-G08-2025/Computer-Vision. [Accessed 2025].

[45] R. Components, "OKdo Nano Developer Kit," [Online]. Available: https://docs.rs-online.com/9149/A700000009238033.pdf. [Accessed 18 February 2025].

[46] Boston Dynamics, "Spot - The Agile Mobile Robot," Boston Dynamics, [Online]. Available: https://bostondynamics.com/products/spot/. [Accessed 20 February 2025].

[47] Boston Dynamics, "Do You Love Me?," 29 December 2020. [Online]. Available: https://www.youtube.com/watch?v=fn3KWM1kuAw. [Accessed 20 February 2025].

[48] NASA, "Mars Exploration Rovers: Spirit and Opportunity," 2 November 2024. [Online]. Available: https://science.nasa.gov/mission/mars-exploration-rovers-spirit-and-opportunity/. [Accessed 22 Febuary 2025].

[49] L. Micheal, "Martian Rover," Thingiverse, [Online]. Available: https://www.thingiverse.com/thing:1318414. [Accessed 22 2 2025].

[50] M. Bagrianski, "Ideas for Your Own (Backward) Mars Rover," Autodesk Indestructables, [Online]. Available: https://www.instructables.com/Ideas-for-Your-Own-Backyard-Mars-Rover/. [Accessed 22 2 2025].

[51] "Raspberry Pi Camera Module 3," [Online]. Available: https://www.pishop.ca/product/raspberry-pi-camera-module-3/. [Accessed 27 February 2025].

[52] "P. J. Reddie, “YOLO: You Only Look Once,”," [Online]. Available: https://pjreddie.com/darknet/yolo/. [Accessed 7 March 2025].

[53] "Wikipedia, “Deep learning,”," [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning. [Accessed 7 March 2025].