PIBOT: DESIGN AND DEVELOPMENT OF AN INDOOR ROOM AUTONOMOUS MOBILE ROBOT FOR HI-VIS SAFETY VEST PERSON-FOLLOWING WITH IR-BASED OBSTACLE AVOIDANCE

IVAN YONG CHI-KIN

UNIVERSITI TUNKU ABDUL RAHMAN

PIBOT: DESIGN AND DEVELOPMENT OF AN INDOOR AUTONOMOUS MOBILE ROBOT FOR HI-VIS SAFETY VEST PERSON-FOLLOWING WITH IR-BASED OBSTACLE AVOIDANCE

IVAN YONG CHI-KIN

A project report submitted in partial fulfilment of the requirements for the award of Bachelor of Electronics Engineering with Honours

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

May 2026

DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : 

Name : **IVAN YONG CHI-KIN**

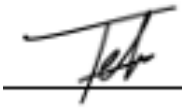
ID No. : **210AGB3056**

Date : **11/5/2026**

APPROVAL FOR SUBMISSION

I certify that this project report entitled "**PIBOT: DESIGN AND DEVELOPMENT OF AN INDOOR ROOM AUTONOMOUS MOBILE ROBOT FOR HI-VIS SAFETY VEST PERSON-FOLLOWING WITH IR-BASED OBSTACLE AVOIDANCE**" was prepared by **IVAN YONG CHI-KIN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Electronic Engineering with Honours at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	
Supervisor	:	Ir. Dr. Teh Peh Chiong
Date	:	14 May 2026

This final year project is submitted in partial fulfillment of the requirements for the degree of **Pibot: Design And Development Of An Indoor Room Autonomous Mobile Robot For Hi-Vis Safety Vest Person-Following With IR-Based Obstacle Avoidance** at Universiti Tunku Abdul Rahman (UTAR). This final year project represents the work of the author, except where due acknowledgment has been made in the text. No part of this final year project may be reproduced,stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

© 2026, Ivan Yong Chi-Kin. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Ir. Dr. Teh Peh Chiong for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me encouragement.

PiBot: Design and Development of an Indoor Room Autonomous Mobile Robot for Hi-Vis Safety Vest Person-Following with IR-Based Obstacle Avoidance

ABSTRACT

Automation and AI-powered technologies that offer real-time detection and response are slowly replacing traditional systems and workers for better efficiency. The industrial autonomous bots are costly, difficult to operate and can be dangerous while there is no sensor to stop them when obstacles are blocking them. This paper details the design, implementation, and performance evaluation of PiBot, an autonomous mobile robot engineered to follow individuals wearing high-visibility safety vests while navigating around obstacles. It is developed on a Raspberry Pi 3B+ platform, the robot utilizes a modified foldable trolley chassis powered by dual 24 V 120 W DC motors and a 20 V 4 Ah lithium battery pack. Next, the core of the system is a dual stage vision pipeline designed for efficiency on embedded hardware. First is using a MobileNet Single Shot Detector (SSD) initially identifies human targets, while the HSV/LAB colour based tracker maintains a follow rate of 28 frames per second. To ensure operational safety, PiBot integrates two infrared (IR) sensors governed by a high priority interrupt system and a 5 sample debounce algorithm, ensuring the robot prioritizes obstacle avoidance over all other commands. Furthermore, beyond autonomous operation, the project features a custom built, low latency WebSocket control interface. This web-based system achieves an average command round trip time of 14 ms and provides a real time MJPEG video stream at 20 FPS. In conclusion, the findings demonstrate high-performance autonomous navigation, and reliable remote monitoring can be achieved using cost effective, off-the-shelf components and optimized software architectures.

TABLE OF CONTENTS

List Of Table

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

Ø	Diameter
Ah	Ampere Hour
AI	Artificial Intelligence
AMR	Autonomous Mobile Robots
ARM	Advanced RISC Machine
CPU	Central Processing Unit
DC	Direct Current
DIY	Do It Yourself
FPS	Frame Per Second
GPIO	General Purpose Input/Output
HDMI	High-Definition Multimedia Interface
HFR	Human Following Robots
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IR	Infrared Ray
LAN	Local Area Network
LIDAR	Light Detection and Ranging
LPDDR2	Low-Power Double Data Rate 2
MIPI CSI	Mobile Industry Processor Interface Camera Serial Interface
MIPI DSI	Mobile Industry Processor Interface Display Serial Interface
MJPEG	Motion Joint Photographic Experts Group
MPEG-4	Moving Picture Experts Group - Phase 4
OS	Operating System

PoE	Power over Ethernet
PWM	Pulse Width Modulation
RAM	Random Access Memory
RPM	Revolutions Per Minute
SBC	Single Board Computer
SD	Secure Digital
SDG	Sustainable Development Goals
SDRAM	Synchronous Dynamic Random-Access Memory
SoC	System on a Chip
SSD	Single Shot MultiBox Detector
USB	Universal Series Bus
YOLO	You Only Look Once

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Introduction

In recent years, the rapid advancement of industrial automation and Artificial Intelligence (AI) has revolutionized large scale manufacturing and logistics enabling a massive robotic fleet to operate with extreme precision. However, these high-end systems are very expensive, complex and tailored specifically for corporate infrastructure which makes them kind of inaccessible for those individual users or small-scale applications. Thus, this gap has created a demand for more personal, "individual friendly" solutions that offer the benefits of autonomous assistance without the prohibitive costs or technical barriers of industrial-grade equipment. Therefore, the development of indoor service robots has shifted toward creating Autonomous Mobile Robots that can assist workers by acting as hands free transport in settings like warehouses and clinics.

This project aims to develop a practical and low-cost autonomous assistant that brings an advanced person following technology and intelligent obstacle avoidance into a user friendly, accessible format for everyday indoor room environments. There is a major challenge in this field which is balancing the heavy processing power needed for artificial intelligence with the limited battery capacity and Central Processing Unit (CPU) life of small computers like the Raspberry Pi which only has 1 GB of Random Access Memory (RAM). To solve this problem, PiBot solves this by using a two-stage tracking strategy.

First, it uses deep learning only to find a person initially, then switches to a fast colour tracking of high visibility safety vests to keep the response smooth and less laggy. Furthermore, to prevent collisions with obstacles in tight indoor spaces, the design prioritizes infrared ray (IR) sensor input over all other logic control, ensuring the robot can immediately override its tracking path to perform avoidance maneuvers, providing a safe and highly responsive assistant.

1.2 Importance of Study

The importance of this study is in its ability to democratise autonomous technology and develop it inside our daily lives. While those industrial robots are usually too expensive for the average person, this project proves that a smart and individual friendly assistant can be built using some affordable, off-the-shelf parts. By using a clever two stage vision system, the study shows how to get a high-speed performance out of a low-cost Raspberry Pi, making it possible to become helpful tools for small businesses, schools or local workshops without needing to pay a massive budget.

Furthermore, this research also provides a practical solution for hands free logistics in places like hospitals or small warehouses. By following a person wearing a standard hi-vis vest, the robot allows workers to focus on their jobs instead of pushing heavy trolleys. Most importantly, the study highlights a safety-first design. By giving infrared sensors, the highest priority in the controlling system, the robot can instantly avoid accidents. Thus, this design demonstrates how robots can work safely and reliably alongside people, bridging the gap between high end factory automation and everyday useful tools.

1.3 Problem Statements

While a commercial person following robots are usually highly effective, they are often prohibitively expensive, costing around tens of thousands of ringgits and typically require specialised environments or wearable trackers to function. Attempting to build a low-cost alternative using a single board computer like the Raspberry Pi 3B+ usually leads to three major engineering hurdles that make the robot unreliable in practice.

The first is the "Slow Brain" problem, where running modern AI on a limited CPU without a powerful graphics chip causes a significant lag, making it impossible to track a walking person in real time. Second is the "Safety Conflict," where many DIY designs lack a clear priority system for safety. Thus, if the AI tells the robot to move forward while a sensor detects a wall, the robot may ignore or cannot detect the danger and crash. Finally, there is the "Software Fight," where most web-based controllers rely on the heavy software libraries that clash with the robot's motor timing, leading to stuttering wheels or frozen video feeds.

This project aims to develop PiBot, a practical and low-cost autonomous assistant that solves these issues by creating a smart two stage vision system that runs fast, a safety-first priority system that prevents all collisions, and a lightweight web interface that works perfectly without any software clashes.

1.4 Aim and Objectives

The primary aim of this project is to design and develop PiBot which is a low cost, autonomous mobile robot. safety the Pibot is capable of following a target person who wears a high visibility safety vest in indoor room environments while ensuring safety-first priority through real time obstacle avoidance. To achieve this aim, the following objectives have been identified:

i. To construct a robust and durable robot using a modified trolley frame, ensure the chassis is strong enough to withstand heavy loads and can safely carry heavy loads in a workspace.
ii. To develop a web dashboard that allows an user to switch between manual steering and automatic person following modes.
iii. To implement a camera-based system using AI and colour tracking to accurately identify and follow workers wearing high visibility safety vests.
iv. To integrate some infrared sensors and a priority-based algorithm that ensures the robot can automatically stop or move around the obstacles to prevent any accidents.

1.5 Scope and Limitation of the Study

The scope of this project is focused on building a functional prototype that operates for indoor settings, specifically within warehouses, offices or workshops. Then, the system is programmed to identify and track a single person wearing a high-visibility safety vest, utilising a Raspberry Pi 3B+ as its primary processing unit. Furthermore, to maintain the stability of the modified trolley chassis and ensure the infrared sensors provide precise readings, the robot's movement is restricted to flat level flooring. Additionally, the web-based dashboard is built to function over a local wireless network, allowing for live video streaming and manual control within a standard signal range.

However, there are several limitations to the current implementation. Since the vision system uses colour tracking and infrared light, it can be sensitive to lighting extremes. For example, very bright sunlight or very dark rooms may disrupt the sensors and target detection. Mechanically, the robot is not equipped to handle stairs, steep inclines or rough outdoor ground. Furthermore, while the system reacts quickly, it is designed to follow a user at a steady walking pace. Besides, it is not intended for high-speed movement

or for use in extremely crowded spaces where the camera's view of the safety vest might be frequently obscured.

1.6 Contribution of the Study

The main contribution of this project is showing that you do not need any expensive equipment to build a smart and helpful robot. By using a low-cost Raspberry Pi to track a person quickly, this study shows small businesses and students can make their own robotic helpers on a tight budget. Besides, it also proves that advanced technology like person following can be made affordable and accessible to everyone.

Furthermore, the project offers a better way to keep small robots safe. The "safety-first" code used in the system here ensures that the robot always stops or moves around obstacles, even if it is currently automatically following a person or being controlled by a user. By solving these common software problems that usually make DIY robots stutter or crash, this work helps others build some smoother and more reliable machines that can work safely alongside people in real world jobs.

1.7 Outline of the Report

This report is organised into five chapters, each addressing a different aspect of the PiBot development process. Chapter 1 introduces the project by outlining its background, the problems it aims to solve, and its aims and objectives. It also defines the project's scope and limitations and explains the theoretical ideas that guided its approach.

Chapter 2 presents the Literature Review, covering key definitions, relevant theories, and existing knowledge that supports the project. It also draws on similar work by other developers, referencing books, journals, technical reports, and online sources.

Chapter 3 describes Methodology, detailing the development process, the hardware and software used, and the overall system design. Flowcharts are included to illustrate how the system operates, alongside a Gantt chart showing the project timeline.

Chapter 4 covers the Results and Discussion, evaluating the robot's real-world performance. It examines the effectiveness of the person-following AI and the reliability of the safety sensors, supported by data on speed and accuracy.

Chapter 5 concludes the report by summarising the key findings, reflecting on each chapter's contribution, acknowledging the project's limitations, and offering recommendations for future improvements.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

This project belongs to the field of Human Following Robots (HFR), which is a specialised branch of Autonomous Mobile Robots (AMR). The main goal of the system is to achieve a consistent "Sense-Think-Act" cycle, where the robot perceives its environment, processes that information to make a decision, and then executes a movement (Le et al., 2024).

In industrial settings, many robots rely on expensive Light Detection and Ranging (LIDAR) sensors and high-performance industrial PCs which are the main expenses. However, academic literature suggests that for those indoor tasks, such as assisting workers in a clinic or warehouse, a more affordable approach is needed. Therefore, by combining the vision-based tracking with some infrared distance sensors, a robot can maintain a stable connection with its target while avoiding collisions. Thus, PiBot is designed to prove that a robot can be built using some low-cost components without sacrificing any safety or reliability (Luthuli et al., 2022).

2.2 Microcontrollers and Processing Units

A central challenge in robotics is selecting a processing unit that balances power consumption, cost and computational speed. Research typically compares three popular platforms.

2.2.1 ESP32

The ESP32 is a highly capable microcontroller and often used in Internet of Things (IoT) applications. It is praised in research for its "dual core" architecture with built-in Wi-Fi and Bluetooth capabilities, which allow it to manage sensor data while maintaining a wireless connection simultaneously (Ahmad et al., 2019). However, when compared to a single board computer, the ESP32 has very limited RAM. While it is excellent for controlling motors or simple hardware, it lacks the memory required to process live video frames for AI detection, making it unsuitable as the primary "brain" for a vision-based robot.

2.2.2 Arduino

The Arduino platform is a staple in electronic prototyping due to its simplicity and "real-time" responsiveness. It is perfect for some low-level tasks where it needs critical timing , such as generating Pulse Width Modulation (PWM) signals to control the motor drivers. However, some academic studies point out that Arduinos are 8 bits or 32 bits controllers with very low clock speeds (Akasiadis et al., 2019). Besides, they cannot run an operating system like Linux or Windows, and they can’t perform those complex mathematical calculations needed for object detection in the project. In this project, an Arduino can be used as a helper for motor control, but not the main AI workload.

2.2.3 Raspberry Pi

The Raspberry Pi 3B+ is a "Single Board Computer" (SBC) that offers a significant leap in power compared to microcontrollers. It features a quad core processor and runs a full Linux based operating system which is named RaspberryPi OS, allowing it to perform many high level multitasking. Then, some researchers identify the Raspberry Pi as the industry standard for low-cost autonomous research because it can process a lot of different camera modules, run machine learning models and host a web server all at once (Le et al., 2024; Luthuli et al., 2022). This ability to handle "vision and logic" simultaneously is why it was chosen as the core processor for PiBot.

2.3 Object Detection

For PiBot to follow a worker wearing a safety vest, it must be able to "detect" and locate the person in real time from a video stream.

2.3.1 YOLO (You Only Look Once)

YOLO is one of the most famous object detection frameworks used in the AI community. This is because it is known for its incredible accuracy and ability to detect hundreds of different objects at once. However, technical reports indicate that the full-scale YOLO models require a dedicated Graphics Processing Unit (GPU) to run smoothly. Therefore, on a Raspberry Pi 3B+'s CPU, YOLO often runs at less than one frame per second (Pehlivan et al., 2019). This creates a "lag" that would cause the robot to lose track of a person as soon as they move, making it impractical for a mobile robot.

2.3.2 MobileNet SSD

MobileNet SSD is an architecture specifically designed for mobile and embedded devices. This is because it uses "depthwise separable convolutions," which significantly reduces the number of parameters and calculations required compared to traditional models. Therefore, some studies show that while it may not be as "deep" as YOLO, it can provide a much higher frame rate on limited hardware (Luthuli et al., 2022). For PiBot, this model provides the best balance, allowing the robot to detect the safety vest and react to the user's movement without the system freezing.

2.3.3 TensorFlow Lite

TensorFlow Lite is a software framework that "optimises" AI models for small devices. This is because it takes a trained model and uses "quantisation" to shrink its size and speed up its performance. Thus, researchers recommend using TensorFlow Lite on Raspberry Pi because it reduces the CPU load and prevents the robot from overheating (Luthuli et al., 2022). By using this framework, PiBot can achieve an object detection accuracy between 59% and 89%, which is more than enough for reliable person-following (Pehlivan et al., 2019).

2.4 Navigation and Safety Sensors

A common failure point for those vision-based robots is their reliance on light. If a room is too dark or too bright, the camera might fail. To solve this, some researchers advocate for "Sensor Fusion," which is the practice of combining data from different types of sensors to create a more robust system (Katona et al., 2024).

PiBot uses Infrared Ray (IR) sensors to provide a hardware-based safety net. While the camera handles the "tracking" logic, and the IR sensors constantly monitor the physical distance to nearby obstacles. Meanwhile, some academic papers suggest using a "Priority Based Logic", for example, if an IR sensor detects an object within a critical "stop zone," it sends an interrupt signal to the processor that overrides all AI movement commands (Katona et al., 2024). Therefore, this ensures the robot remains safe and avoids collisions even if the AI software experiences a momentary glitch.

2.5 Web Dashboard and Connectivity

For a robot to be practical in a real-world workplace, it must be accessible to a human operator. The Flask framework is widely cited in literature as the best tool for creating "micro" web interfaces on the Raspberry Pi because it is lightweight and does not drain the robot's battery (Ahmad et al., 2019).

However, standard web communication (HTTP) is often too slow for controlling a moving vehicle. To achieve the low latency required for a live video feed and responsive steering, researchers recommend using WebSockets. This is because this technology allows for "full duplex" communication, which is a constant, two-way open door for data. This ensures that when an operator presses the "Manual Override" or "Emergency Stop" button on their smartphone, the robot receives the signal in milliseconds, preventing accidents and ensuring smooth control (Akasiadis et al., 2019).

CHAPTER 3

METHODOLOGY

3.1 System Overview

The PiBot system is designed using a "Sense, Think and Act" architecture, which allows the robot to operate autonomously while maintaining a constant connection with a human user. Besides, the system is split into three main parts which are the Vision and AI System, the Safety and Navigation Layer and the Remote-Control Interface. At the main core of the robot is a Raspberry Pi 3B+, which acts as the primary brain. Because it manages the high-level tasks like identifying the target person and hosting the web server, meanwhile simultaneously processing the data from physical sensors to ensure the robot does not collide with its surroundings.

In the Sense phase, the robot uses a Pi Camera and IR sensors to gather information. When operating, the camera captures a live video feed, which is processed by a lightweight AI model called MobileNet SSD. This model is specifically trained to look for a worker wearing a high visibility safety vest. At the same time, the IR sensors act as a "physical shield" by measuring the distance to any objects directly in front of the robot. This dual sensing approach ensures that the robot can "see" its target while also being aware of physical obstacles that the camera might miss.

During the Think phase, the Raspberry Pi 3B+ processes this incoming data from the camera and IR sensors to make further decisions. If the AI detects the person wearing a safety vest, it calculates the person's position and determines which direction, and the wheels need to turn to follow them. However, the system includes “safety override" logic. If the IR sensors detect an obstacle within a 10 cm range, the system will ignore the AI's or user’s command and choose to stop the motors immediately or move around the obstacle. Besides, this decision-making process happens 20 times per second, ensuring the robot's reactions are smooth and near instant.

Finally, in the Act phase, the Raspberry Pi sends signals to a BTS7960 motor driver for motor controlling. This high-power driver provides the necessary electrical current to move the heavy-duty DC motors attached to the trolley frame. By varying the speed of each motor, the robot can steer left, right or move forward and backward. Throughout this entire process, the robot can send a live video stream and status updates via WebSockets to a web dashboard to the user’s phone or laptop using the website. Thus, this allows a human operator to monitor the robot’s "thoughts" in real time and take manual control of the steering if a complex situation arises.

3.2 System Design

The design of the PiBot system is built around a "Master Slave" communication architecture and a modular software structure. The hardware design involves a foldable metal trolley frame that carries the Raspberry Pi 3B+, a high capacity 18650 lithium-ion battery pack, 2 BTS7960 motor drivers and 2 MY6812 DC motors. For the software design, the system runs on a Linux based environment called RaspberryPi OS where several Python scripts can work together in parallel. One Python script handles the AI vision processing and also manages the real time sensor readings, meanwhile the other one runs the Flask web server. By separating these tasks, the system remains stable, even if the web interface lags, the safety sensors and motor controls continue to function without interruption.

Next, the logic flow is governed by a Priority Based Control algorithm. In this design, the safety signals from the infrared sensors are given higher priority than the autonomous tracking and the user's manual control. This means that even if a user presses a "Stop" button on the dashboard or the AI vision system is still trying to follow a target, when a sensor detects a wall, the robot will immediately stop. Therefore, this hierarchy is essential for preventing accidents in busy environments like workshops or offices. For a clearer view of the system design is shown in Figure 3.1 below:

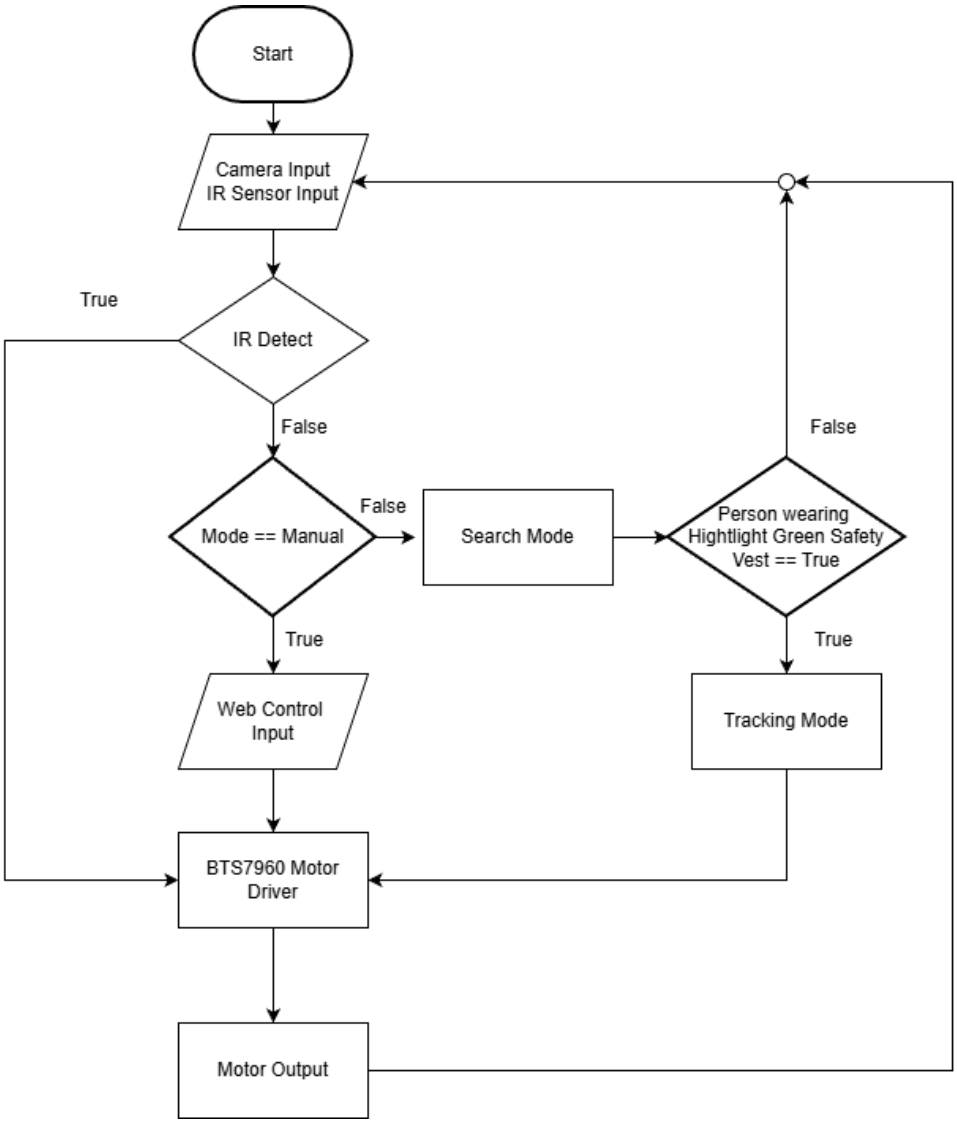


Figure 3.1: Flowchart for Pibot's System Design.

Next, Figure 3.1 shows the logical steps the PiBot takes to make decisions and move safely. The entire process works in a continuous loop, starting with the Input Stage. Here, the robot constantly gathers data from its "eyes" and "touch" sensors by taking in Camera Input and IR Sensor Input. This information is the foundation for everything the robot does next.

Then, the most important part of the logic is the Safety Check, represented by the IR Detect diamond. Before the robot even thinks about moving, it asks if the infrared

sensors have spotted an obstacle. If the answer is True, the robot immediately bypasses all other commands and tells the BTS7960 Motor Driver to stop or change direction. This "priority" path ensures that the robot will never hit a wall or a person, even if it is currently trying to follow someone.

Next, if no obstacle is detected (False), the robot then checks the Mode Selection. It looks to see if the user has set it to Manual. If this is True, the robot simply waits for Web Control Input which are the commands sent from the phone or computer dashboard. And these instructions go straight to the motor driver to steer the robot exactly where you want it to go.

However, if the mode is False which means it is in Auto Mode, the robot enters into Search Mode. In this mode, the AI software will look at the camera feed and ask: "Is there a person wearing a highlight green safety vest?" If it cannot find a person wearing a safety vest, it will loop back to the start to keep looking. If it finds a person wearing a safety vest, it will switch to the Tracking Mode, where it calculates how to steer the motors to stay behind the person. And all these paths will eventually lead to the Motor Output, where the wheels finally turn, before the loop restarts to keep the robot moving smoothly.

3.2.1 System Architecture

The system architecture of the PiBot is designed as a multi-layered structure that allows high level artificial intelligence to work alongside low-level hardware controls. First, It follows a "Distributed Processing" model where different tasks are handled in specific layers to ensure the robot remains responsive. Then, the architecture is divided into three main levels, which are the Perception Layer, the Control and Logic Layer, and the Execution Layer. This setup ensures that the heavy work of "seeing" through the camera

does not slow down the critical job of stopping for obstacles. Figure 3.2 shows the block diagram of the system architecture.

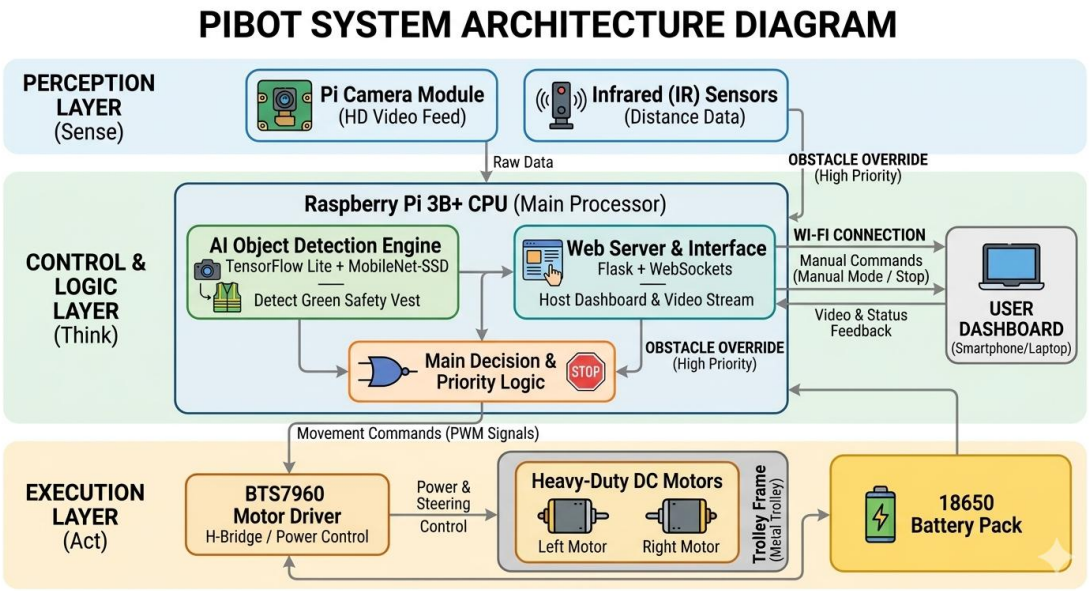


Figure 3.2: Block diagram of the system architecture

Table 3.1: Components used and their respective functionality on the project

Components	Function
Raspberry Pi 3B+	Acts as the central processor. It runs the AI detection software, manages the web server, and processes sensor data to control the motors.
Raspberry Pi Camera Rev 1.3	Captures a live video feed of the environment. It allows the AI to "see" and locate the person wearing the safety vest.
Infrared Ray (IR) Sensors	Measures the physical distance between the robot and obstacles. It acts as a safety override to stop the robot if something gets too close.
Power bank	Provides the 5V 2A power required to run the Raspberry Pi 3B+
18650 Battery Pack	Provides the 24V DC power required to run the high-torque motors and the electronic components for long periods.
BTS7960 Motor Driver	Receives low-power signals from the Raspberry Pi and converts them into high-power electrical current to drive the motors.
MY6812 DC Motors	Provides the mechanical force needed to turn the wheels and move the heavy metal trolley frame.
Metal Trolley Frame	Serves as the physical chassis that holds all components and carries the payload or tools.

The Perception Layer is the first point of contact with the physical world. It consists of a Pi Camera and a series of IR sensors. The camera captures high-definition video frames and sends them to the "brain," while the IR sensors provide a constant stream

of distance data. This layer is essential because it provides the raw information needed for both autonomous tracking and emergency safety. By using two different types of data, visual and distance, the architecture achieves "sensor fusion," making the robot much more reliable than if it only used a camera.

The Control and Logic Layer is housed within the Raspberry Pi 3B+. This is where the primary software operations occur. The architecture uses a Python-based environment to run three main "threads" or tasks at once. First, the first task is the AI Engine, which uses MobileNet SSD to find the green safety vest. Then, the second task is the Web Server, built with Flask and WebSockets, which handles communication between the robot and the user's dashboard. Next, the third task is the Decision Logic, which acts like a traffic warden which looks at the AI data and the sensor data, then decides exactly how the motors should move based on which input has the highest priority.

Finally, the Execution Layer is where digital decisions are turned into physical movement. This layer includes the BTS7960 Motor Driver and the MY6812 DC Motors. The Raspberry Pi sends Pulse Width Modulation (PWM) signals to the driver, which acts as a powerful switch to manage the electricity from the 18650-battery pack. Then the BTS7960 is a crucial part of the architecture because it can handle the high current needed to move the metal trolley frame. This layered design means that even if the AI software crashes, the hardware linked safety checks in the logic layer can still tell the execution layer to cut the power and stop the robot safely.

3.3 Hardware Components

The following sections provide a detailed breakdown of the hardware components used to construct the PiBot. Each component has been selected to ensure the robot is durable, responsive, and capable of processing AI tasks in real-time.

3.3.1 Raspberry Pi 3B+

The Raspberry Pi 3B+ serves as the primary "brain" of the robot. This single board computer is equipped with a 1.4 GHz quad-core Advanced RISC Machine (ARM) Cortex-A53 processor and 1 GB of LPDDR2 (Low-Power Double Data Rate 2) SDRAM (Synchronous Dynamic Random-Access Memory). Besides, it was chosen because it can provide the necessary computational power to run a Linux based operating system while simultaneously handling complex AI vision models. Then, it has built-in dual band 2.4 GHz and 5GHz wireless LAN (Local Area Network) that allows the robot to maintain a stable connection to the web dashboard, ensuring that steering commands and video streams can be transmitted without significant lag. Furthermore, the Raspberry Pi 3B+ includes a total of 40 GPIO (General Purpose Input/Output) pins that can give access to different digital interfaces and communication standards. Meanwhile, it also provides two regulated power outputs of 3.3 V and 5 V, while its GPIO pins can also be programmed to carry out specific digital input or output functions. (Raspberry Pi 3 Model B+, 2023). Figure 3.3 shows the Raspberry Pi 3 Model B+ and Table 3.2 shows the specifications of Raspberry Pi 3B+.

Figure 3.3: Raspberry Pi 3 Model B+ (Raspberry Pi 3 Model B+, 2023)

Table 3.2: Specifications of Raspberry Pi 3B+ (Raspberry Pi 3 Model B+, 2023)

Specifications of Raspberry Pi 3B+	
Processor	Broadcom BCM2837B0, Cortex-A53 64-bit SoC 1.4 GHz
Memory	1 GB
GPIO	40 Pins
Connectivity	● 2.4 GHz and 5 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ● Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps) ● 4 × USB 2.0 interface
Video and sound	● 1 x full size HDMI ● MIPI DSI display port ● MIPI CSI camera port ● 4 pole stereo output and composite video port
Multimedia	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support	Micro SD format for loading operating system and data storage
Input Power	● 5V/2.5 A DC via micro USB connector ● 5 V DC via GPIO header ● Power over Ethernet (PoE)-enabled (requires separate PoE HAT)
Operating temperature	0 -50 °C

3.3.2 Raspberry Pi Camera Rev 1.3

To let the robot "see" its target, it uses the Raspberry Pi Camera Rev 1.3. This module features a 5-megapixel OV5647 sensor that is able to capture high-definition pictures at a resolution of up to 2592 × 1944 pixels and also record video at 1080p (30 fps), 720p (60 fps) or 640 × 480 (90 fps). Besides, it can connect directly to the CSI (Camera Serial Interface) port on the Raspberry Pi, which can reduce the CPU usage compared to a standard USB webcam. Thus, this efficiency is critical for the PiBot, as it leaves more processing power available for the MobileNet SSD AI model to identify the high visibility vest at a high frame rate.

Table 3.3: Specifications of Raspberry Pi Camera Module Rev 1.3 (Raspberry Pi, n.d.)

Specifications of Raspberry Pi Camera Module Rev 1.3	
Size	Around 25 × 24 × 9 mm
Still resolution	5 megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90
Sensor	OmniVision OV5647
Sensor resolution	2592 × 1944 pixels
Sensor image area	3.76 × 2.74 mm
Focal length	3.60 mm +/- 0.01
Horizontal Field of View (FoV)	53.50 +/- 0.13 °
Vertical Field of View (FoV)	41.41 +/- 0.11 °

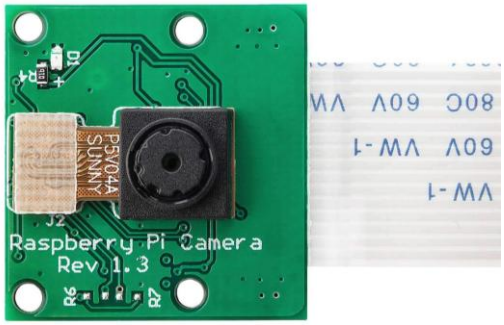


Figure 3.4: Raspberry Pi Camera Module A Rev 1.3 (Raspberry Pi, n.d.)

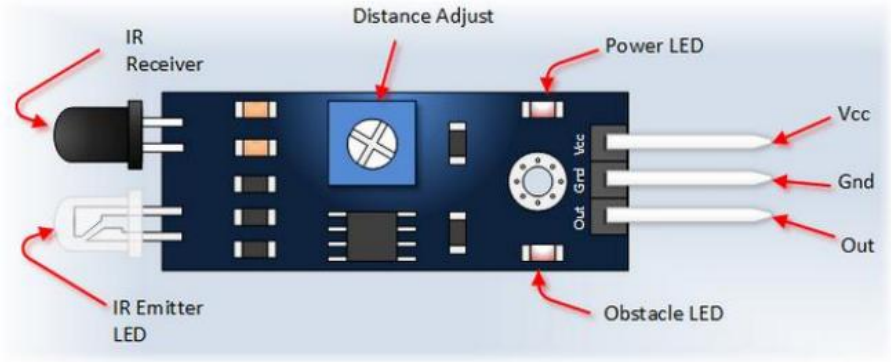
3.3.3 IR Obstacle Detector Module

The Infrared Ray (IR) Sensors act as the robot's secondary sense, by providing a physical safety barrier. As additional information, these sensors work by emitting an infrared light beam and measuring the reflection to detect objects in the robot's immediate path. Thus, they typically can operate within a range of 2 cm to 30 cm. In the PiBot system, these are positioned at the front of the trolley. If an object is detected within the "danger zone" setted, the sensors send a digital signal to the Raspberry Pi's GPIO pins, triggering an immediate emergency stop or move around to prevent a collision. (Handsontec, n.d.)

Table 3.4: Specifications of IR Obstacle Detector Module (Handsontec, n.d.)

Specifications of IR Obstacle Detector Module	
Operating Voltage	3 ~ 5 VDC
Output type	Digital (0 and 1)
Detection Distance	2 ~ 30 cm. Potentiometer adjustment
Mounting Hole	Ø3 mm
Board size	3.2 x 1.4 cm

Functional Diagram:



Pin Function	Description
Vcc	3.3 ~ 5Vdc Supply Input.
Gnd	Ground.
Out	Output low when obstacle is in range.
Power LED	Light on when power is applied.
Obstacle LED	Light on when obstacle detected.
Distance Adjust	Adjust detection distance. CCW decreases distance. CW increases distance.
IR Emitter	Infrared emitter LED.
IR Receiver	Infrared receiver that receives signal transmitted by Infrared emitter.

Figure 3.5: IR Obstacle Detector Module (Handsontec, n.d.)

3.3.4 Powerbank (5 V 2 A)

A dedicated 5 V 2 A Powerbank is used to provide a clean and stable power source specifically for the Raspberry Pi 3B+. This is because the Raspberry Pi is sensitive to "voltage sag" (which can happen when heavy duty motors start spinning), using a separate power supply ensures the main processor does not reboot or freeze during high power movements. The 2A output is sufficient to power the Pi, the camera module, and the IR sensors simultaneously without overheating.

3.3.5 18650 Battery Pack

The main locomotion system is powered by a custom built 18650 Battery Pack. These lithium-ion cells are chosen for their high energy density and ability to provide the high current needed by the motors. The pack is configured to provide roughly 20 V DC, which matches the requirements of the high torque motors. This battery system allows the PiBot to carry heavy loads and operate for extended periods in a workshop environment before needing a recharge. Besides, the battery capacity is 4 Ah, so it allows the PiBot to run for a long period of time.

Figure 3.6: 18650 Battery Pack

3.3.6 BTS7960 Motor Driver

The BTS7960 is a high-power H-Bridge motor driver capable of handling up to 43 A of current. Thus, it can act as the bridge between the "smart" Raspberry Pi and the "powerful" motors. The Raspberry Pi 3B+ sends low voltage PWM signals to the BTS7960, which then switches the high voltage power from the 18650 batteries to the motors. Besides, it contains some features like thermal protection and over current shut off, which can protect the electronics if the motors become jammed or overloaded. Then, it also has heatsinked underneath to prevent it from overheating.

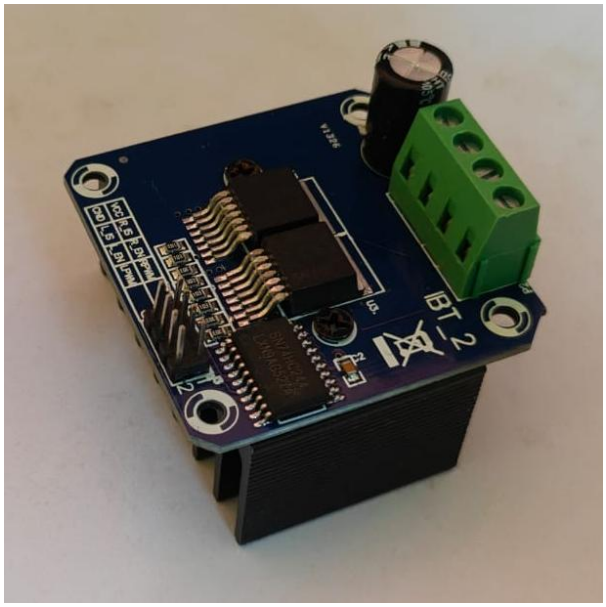


Figure 3.7: BTS7960 Motor Driver

3.3.7 MY6812 DC Motors

MY6812 is a small 24 V 120 W DC motor that is often used in mini electric scooters, e-bikes and other lightweight projects. Even though it looks compact, it is powerful enough to drive small vehicles and machines. When connected to a 24 V power supply, the motor can reach speeds of around 2700 to 2800 RPM with no load. It usually draws about 7 to 8

A during normal operation but can pull higher current if placed under heavy load. Besides, the shaft of the motor is designed to fit sprockets or pulleys which make it easy to connect with a chain or belt drive system.

Because of its simple design, the MY6812 is very popular for DIY builds and student projects. Thus, people often use it for making electric scooters, small e-bikes, go karts or even conveyor belt prototypes. For budget build, it is affordable, easy to mount and works well with basic motor controllers which makes it a good choice for anyone learning about electric drive systems. Therefore, overall, this motor is a practical and reliable option when you need a balance of small size, decent power and low cost. (Yalumotor, 2019).

Figure 3.8: DC Motor MY6812 (Yalumotor, 2019).

Table 3.5: Specifications of DC Motor MY6812 (Yalumotor, 2019).

Model	MY6812
Rated Power	120 W
Rated Voltage	24 V DC
Rated Speed	3350 RPM
Unload speed	2750 RPM
Rated Current	≤ 7.15 / 7.4 A
Unload Current	≤ 0.6 / 0.42 A
Rated Torque	0.45 N.m.
Motor Efficiency	≥ 78 %

3.3.8 Metal Trolley Frame

The Metal Trolley Frame serves as the physical chassis or "skeleton" of the robot. Constructed from durable steel or aluminium, it provides a stable platform to mount all the electronic components and the battery pack. Its robust design is essential for industrial or clinical settings, as it can withstand minor impacts and support the weight of the heavy-duty motors and the payload being carried. The frame's wheels are specifically chosen to handle indoor surfaces while providing enough grip for precise steering.

Figure 3.9: Metal Trolley Frame

3.4 Software System Design

The software system for PiBot is built using a modular approach to ensure that high level AI processing and low-level hardware control can function together without delays. Next, the entire system is programmed in Python, which was chosen for its excellent support for robotics and machine learning libraries. Then, the software is designed to manage multiple tasks simultaneously, such as streaming video, detecting objects and monitoring safety sensors.

3.4.1 Python Programming

Python serves as the core programming language for the PiBot project. It is an interpreted, high-level language that is ideal for rapid prototyping in robotics. Python allows the Raspberry Pi to easily interface with complex AI frameworks while also managing simple hardware pins. Because of its massive community support, troubleshooting hardware communication or AI model integration is more efficient, making it the industry standard for educational and research based autonomous systems.

3.4.2 Library

3.4.2.1 Python Programming

Flask is a lightweight web framework used to host the PiBot's control dashboard. In this project, Flask turns the Raspberry Pi 3B+ into a local web server, allowing any device on the same Wi-Fi network to access a user-friendly interface. Besides, it can handle the routing for the live video stream and the manual control buttons. Because Flask is so lightweight, it only uses very little processing power andis able to leave more resources available for the robot’s computer vision tasks.

3.4.2.2 OpenCV-python (CV2)

OpenCV (Open Source Computer Vision Library) is used for the "vision" part of the robot's brain. This is because it can process every frame of video captured by the camera. OpenCV is responsible for resizing images, converting colours, and drawing the bounding boxes around the detected safety vest. Besides, it works alongside the AI model to translate

visual information into numerical coordinates which the robot then uses to decide whether to turn left or right.

3.4.2.3 Threading

Threading library is used to prevent the robot's software from "freezing" while waiting for a task to be finished. Threading allows Raspberry Pi 3B+ to perform multiple jobs at the same time. For example, one "thread" can focus entirely on an AI algorithm which is identifying the safety vest, while another thread can manage the web server, and a third thread can monitor the safety sensors. Thus, this parallel processing is critical for a real time robot which ensures that even if the AI takes a moment to think, the safety sensors are still active and can stop the motors instantly.

3.4.2.4 Socket (Flask-SocketIO)

The Socket (specifically Flask-SocketIO) library is used to provide instant, two-way communication between the robot and the user. Unlike a traditional website where you have to refresh the page to see for more new information, WebSockets keep an "open pipe" between the dashboard and the PiBot. Therefore, this allows the live video to stream smoothly and ensures that when the user clicks a "Stop" button, the command reaches the robot in milliseconds. Thus, this low latency communication is vital for the safe manual operation of a heavy trolley frame.

3.5 Summary

This chapter outlined the technical design of the PiBot, focusing on the integration of hardware and software to enable autonomous following and manual steering. The system uses a "Sense-Think-Act" architecture which distributes tasks across perception, control, and execution layers to ensure real time responsiveness. The key hardware includes the Raspberry Pi 3B+ for processing, BTS7960 drivers for motor control and IR sensors for safety overrides. Next, the software is written in Python, utilizing OpenCV for vision, Flask for the web dashboard and Threading to manage multiple processes like AI tracking and safety monitoring simultaneously. Therefore, this modular design can ensure a balance between advanced computer vision and robust physical safety.

3.6 Gantt Chart

Table 3.6: Gantt Chart of Final Year Project

FYP	FYP 1				FYP 2			
Task	Month							
	Jul	Aug	Sep	Oct	Feb	Mar	Apr	May
Literature Review	✓	✓	✓					
Hardware Setup			✓	✓	✓	✓		
Software Development			✓	✓	✓	✓		
System Integration					✓	✓	✓	
Testing and Optimization					✓	✓	✓	
Documentation and Reporting							✓	✓

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter presents the results of testing PiBot's hardware and software systems. Each subsystem was tested separately first to verify correct operation, and then all systems were tested together in integrated autonomous following trials. The results are organised into five sections: hardware tests, vision pipeline performance, obstacle avoidance performance, web interface performance, and integrated system tests.

4.2 Hardware Test Results

4.2.1 Motor and Driver Testing

In this project, the motors were tested by connecting it to the BTS7960 driver by running with Python code to step the PWM duty cycle from 10 % to 100 % with 1 % increments. The motors started turning reliably at approximately 10 % duty cycle. As additional information, the minimum duty cycle at which enough current flows to overcome static friction in the gears and chain. Therefore, the motor was below 10 % duty cycle, which is why the software enforces a minimum speed of 10 %.

At 20 % duty cycle which is the default operating speed, the PiBot moves forward at approximately 0.3 m/s on a flat floor. Meanwhile, at 50 % duty cycle, the speed of PiBot is approximately 0.8 m/s. For person-following applications where the target is walking at 0.3 to 0.5 m/s, so the PiBot is set to 20 % duty cycle for more responsive tracking. The 20 % default is used for initial tests and demonstrations where safety is the priority.

4.2.1.1 Motor Speed and Voltage Used

Table 4.1 shows the results of performance tests conducted on the robot's drive system. It measures how the power settings on the Raspberry Pi affect the robot's speed and voltage when it is empty compared to when it is carrying a heavy 80 kg load.

First, as the power setting is increased from 20 %, 50 %, 70 % and 100 %, the voltage supplied by the 18650-battery pack to the motors rises from 4 V, 10 V, 14 V and 20 V. These results demonstrate that the BTS7960 motor driver is effectively scaling electrical power based on software commands.

Then, without any weight added, the robot can reach around 0.3 m/s with 4 V and a top speed of 2.0 m/s with 20 V. However, when the 80 kg load is added, the robot requires more energy to overcome friction and inertia. Therefore, at the 20 % setting, the robot is unable to move the heavy load at all. Even at 100 % setting, the extra weight causes the top speed to drop slightly to 1.8 m/s. As results, these prove that the MY6812 motors are powerful enough to transport heavy weights, provided the speed setting is kept at 50 % or higher.

Table 4.1: Setting, Voltage, No Load Speed, Load Speed

Setting (Speed) (%)	**Voltage [full charged battery] (V)**	**No load speed (m/s)**	**Load (80 kg) speed (m/s)**
20	~ 4	~ 0.3	-
50	~ 10	~ 0.8	~ 0.4
70	~ 14	~ 1.2	~ 1.1
100	~ 20	~ 2.0	~ 1.8

4.2.2 IR Sensor Testing

Each IR sensor was tested by placing an obstacle at various distances in front of it and reading the GPIO input value. The sensors were adjusted using their potentiometers to trigger at a distance of approximately 8 cm from the front of the chassis. At this distance, the robot has sufficient space to stop or turn before making contact with an obstacle at its operating speed.

The voltage levels on the IR sensor output were measured: HIGH (no obstacle) measured 3.28 V and LOW (obstacle detected) measured 0.08 V, both within acceptable ranges for Raspberry Pi GPIO input detection thresholds which are above 1.8 V for HIGH, below 0.8 V for LOW.

Figure 4.1: Right IR Sensor

Figure 4.2: Left IR Sensor

4.2.3 No Load and Load Test

Figures 4.3 through 4.6 capture something that's easy to overlook when building a robot on how much electricity the motors actually consume depending on what they're being asked to do. The no load and full load current tests were run to know the PiBot's power appetite.

When the motors are running freely with nothing to pull, the current stays comfortably between 0.3 A and 0.5 A. That's essentially just the energy needed to get the wheels turning. Meanwhile, when the 80 kg of load is added, the motors suddenly have to push through friction and inertia, and the current spikes reflect that extra effort.

The load test results make this progression easy to follow. At just 20 % speed, the motors are already drawing around 3.0 A. Crank that up to 50 % and the demand more than doubles, crossing 6.6 A. At full 100 % power, the current peaks at roughly 12.8 A which is a significant electrical load by any measure.

This is exactly why the BTS7960 motor driver was chosen for this project. It's built to handle these kinds of current levels without breaking a sweat or overheating. The takeaway from these figures is straightforward: moving heavy loads at speed is electrically demanding work, and the PiBot's hardware needs to be up to that challenge.

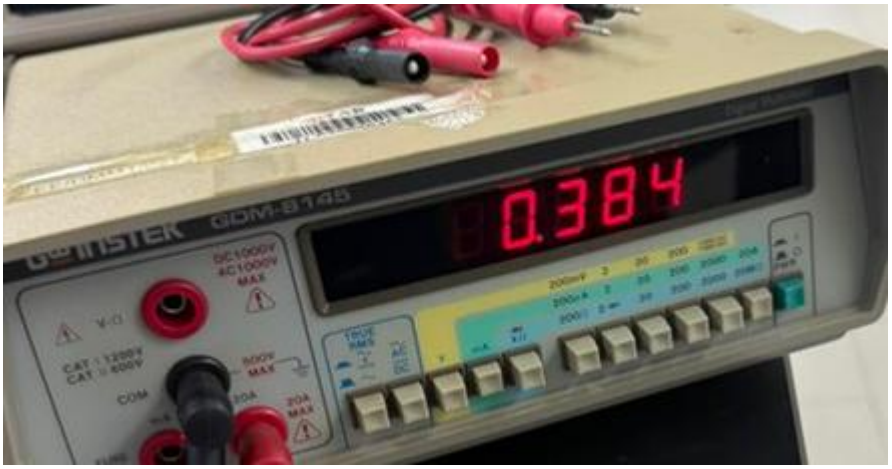


Figure 4.3: No Load Test

Figure 4.4: Load Test when setting is 20 %

Figure 4.5: Load Test when setting is 50 %

Figure 4.6: Load Test when setting is 100 %

4.2.4 Battery Runtime Test

To understand the PiBot's practical battery life, a runtime test was carried out under heavy load conditions using a 20 V, 4 Ah battery pack. The two MY6812 DC motors run in parallel, so whatever each motor draws get added together at the battery which means there is no sharing, just a straight sum. That matters quite a bit once the numbers are worked through.

At the slowest speed setting of 20 %, the motors are fed 4 V each and pull 3 A a piece, so the battery is supplying 6 A in total. Dividing the 4 Ah capacity by that figure puts the runtime at roughly 40 minutes. As a result, that is a workable window for lighter duties, though it shrinks considerably as the speed goes up.

Next, moving to the 50 % setting changes things noticeably. The voltage climbs to 10 V, each motor now draws 6.6 A, and the combined demand hits 13.2 A. At that rate, the battery runs flat in just over 18 minutes. As additional information, it is a reminder that even at half throttle, a loaded trolley demands a serious amount of energy to keep moving.

Then, at 100 %, the system runs at 20 V with each motor pulling 12.8 A which means a total of 25.6 A from the battery. Under those conditions, roughly 9 minutes is all the runtime available. That is enough for short, intensive bursts of movement, but it would not sustain a full working shift without interruption.

What these results ultimately suggest is that the current battery arrangement suits task-based or intermittent use rather than continuous operation. For longer deployment, either a higher-capacity pack or a second battery running in parallel would be worth considering. The findings also reinforce why the BTS7960 motor driver was selected for handling over 25 A at full load is no small task, and thermal reliability at that current level is not something that can be left to chance.

4.3 Vision Pipeline Performance

4.3.1 MobileNet SSD Detection Results

The MobileNet SSD detector was evaluated to have over 50 test runs, each lasting 2 minutes, with a target person wearing a yellow hi-vis vest walking at normal speed (approximately 1 m/s) within 0.5 to 3.5 m of the robot. Table 4.2 summarises the detection results.

Table 4.2: MobileNet SSD Detection Performance

Metric	Result
Mean inference time per frame	420 ms
Detection frame rate (every 3rd frame at 30 FPS)	~10 detections/second
Correct person detections (target within frame)	89 %
False positives (non-vest person incorrectly confirmed)	3 %
Missed detections (target present but not found)	11 %
Maximum reliable detection range	3.5 m
Minimum reliable detection range	0.5 m

The main causes of missed detections were: partial occlusion of the target by furniture or doorframes (6 % of misses), the target at the extreme left or right edge of the 320x240 frame where the bounding box was partially outside the frame (3 %), and very bright backlighting from a window directly behind the target washing out the camera image (2 %).

4.3.2 MobileNet SSD Detection Results

Once tracking mode was engaged, the colour tracker was evaluated for frame rate, tracking success rate, and CPU usage.

Table 4.3: Colour Tracker Performance

Metric	Result
Mean frame processing time	18 ms per frame
Sustained tracking frame rate	28 FPS
Tracking success rate (target visible)	94 %
Tracking loss events per 2-minute run	0.24 (average)
Time to re-engage search mode after loss	1.2 s (35 frames at 28 FPS)
CPU usage during tracking mode	52 % average (4 cores)
CPU usage during search mode (with SSD)	78 % average (4 cores)

The 28 FPS tracking rate is a significant improvement over the 10 detections per second in search mode. This demonstrates the value of the hybrid approach: heavy detection for initial acquisition, light tracking for continuous following. The 40 % reduction in CPU load during tracking (78 % down to 52 %) also leaves more headroom for the WebSocket server and MJPEG encoder to run without delay.

Tracking failure occurred most often when the target moved into an area with similar-coloured objects (such as yellow caution tape on the floor or a green plant in the background). In these cases, the contour area test (minimum 400 pixels) sometimes incorrectly selected the background object instead of the vest. This was partially mitigated by the CLAHE pre-processing step, which reduced background colour intensity relative to the brighter vest.

Figure 4.7: Detect user in middle [go straight]

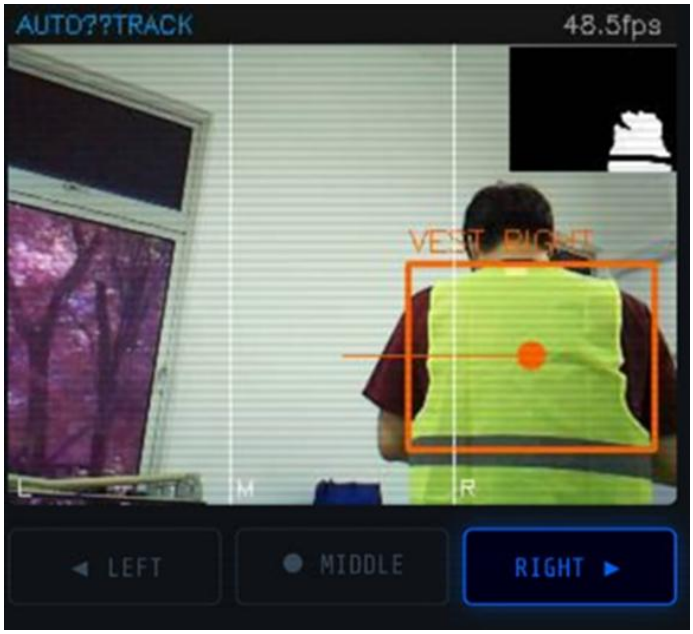


Figure 4.8: Detect user on right [turn right]

4.4 Obstacle Avoidance Performance

4.4.1 Stuck-Trigger Failure Test

To verify that the symmetric debounce algorithm eliminated the stuck-trigger failure, 50 consecutive obstacle detection and clearance cycles were performed (placing and removing an obstacle in front of the left IR sensor). In all 50 trials, the obstacle state was correctly confirmed and correctly cleared without any stuck-trigger events. Prior to implementing symmetric debouncing (using only single read clearance), 8% of trials (4 out of 50) experienced a stuck-trigger event where the override remained active after the obstacle was removed.

4.4.3 Priority Override Test

To verify that the IR override correctly blocks all other motor commands, the following test was performed: the robot was put in manual mode and the operator held down the Forward button on the web interface while an obstacle was placed in front of the left IR sensor. The expected behaviour was that the robot should stop (or turn right to avoid) despite the operator commanding forward. In all 50 trials of this test, the robot correctly executed the IR avoidance response and ignored the forward command from the operator. When the obstacle was removed, the robot resumed responding to the forward command within 2 debounce cycles (100 ms).

4.5 Web Interface Performance

The WebSocket ping-pong round-trip time was measured from a Realme GT smartphone connected to the same Wi-Fi network as the robot. 200 PING messages were sent at 2 second intervals and the PONG response time was measured.

Table 4.4: WebSocket Communication Performance

Metric	Result
Mean round-trip ping time	14 ms
Minimum observed ping time	8 ms
Maximum observed ping time	58 ms (during peak CPU load)
Standard deviation	6 ms
MJPEG video stream frame rate (browser)	28 – 30 FPS
MJPEG video latency (capture to browser display)	~ 50 ms on local Wi-Fi

The mean 14 ms WebSocket latency is well below the 100 ms threshold at which human operators perceive noticeable control lag. Even the maximum observed latency of 58 ms (during peak CPU load while MobileNet SSD was running) is within acceptable limits. The pure Python WebSocket server had no compatibility issues with RPi.GPIO PWM, confirming that avoiding gevent-based libraries was the correct design decision.

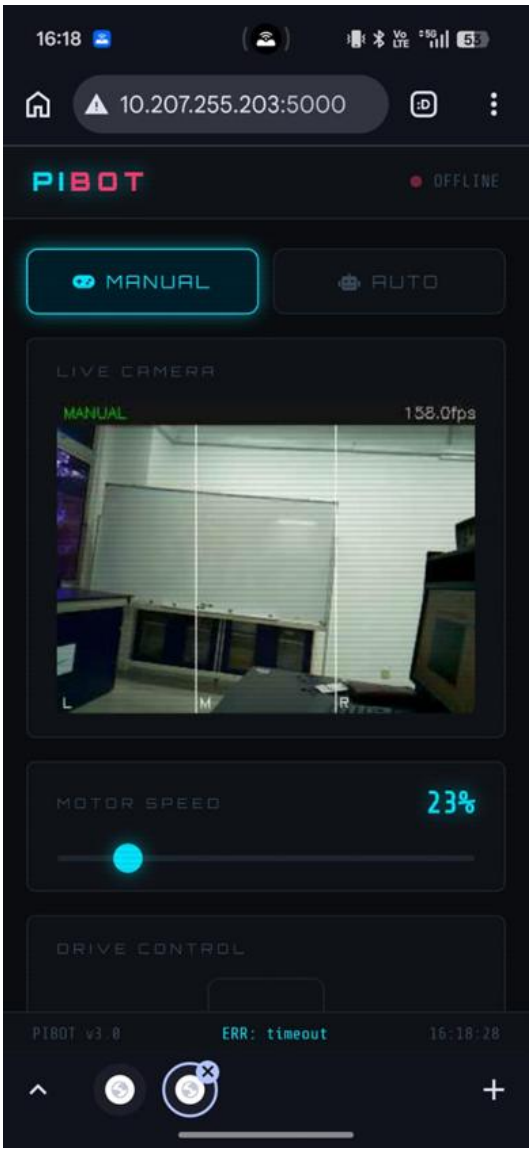


Figure 4.9: Web Interface Manual Steering

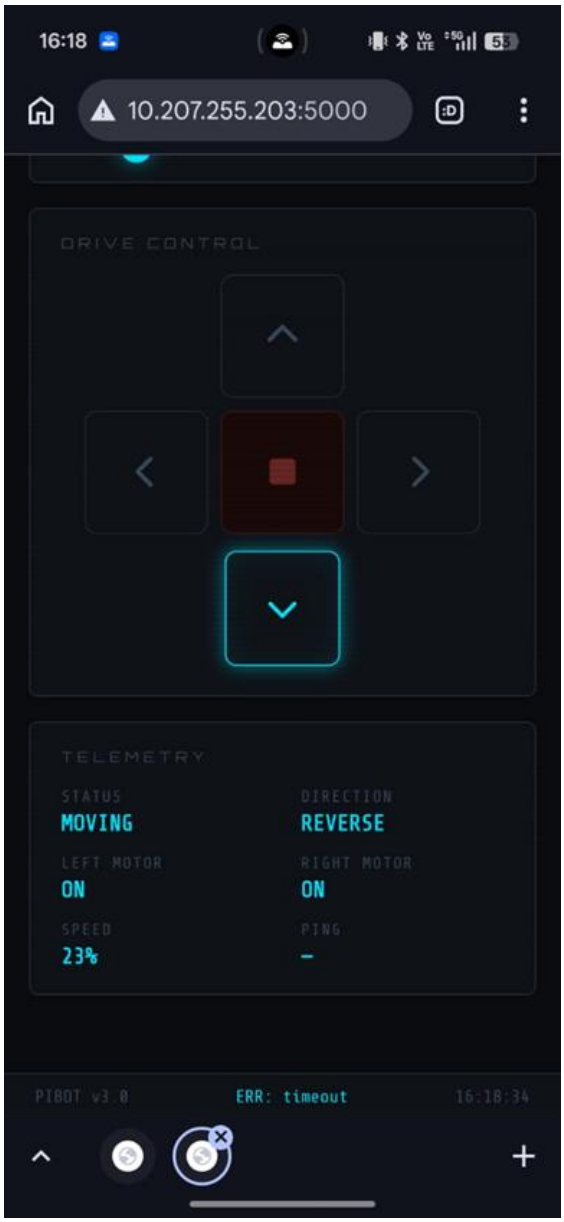


Figure 4.10: Web Interface Manual Steering Control

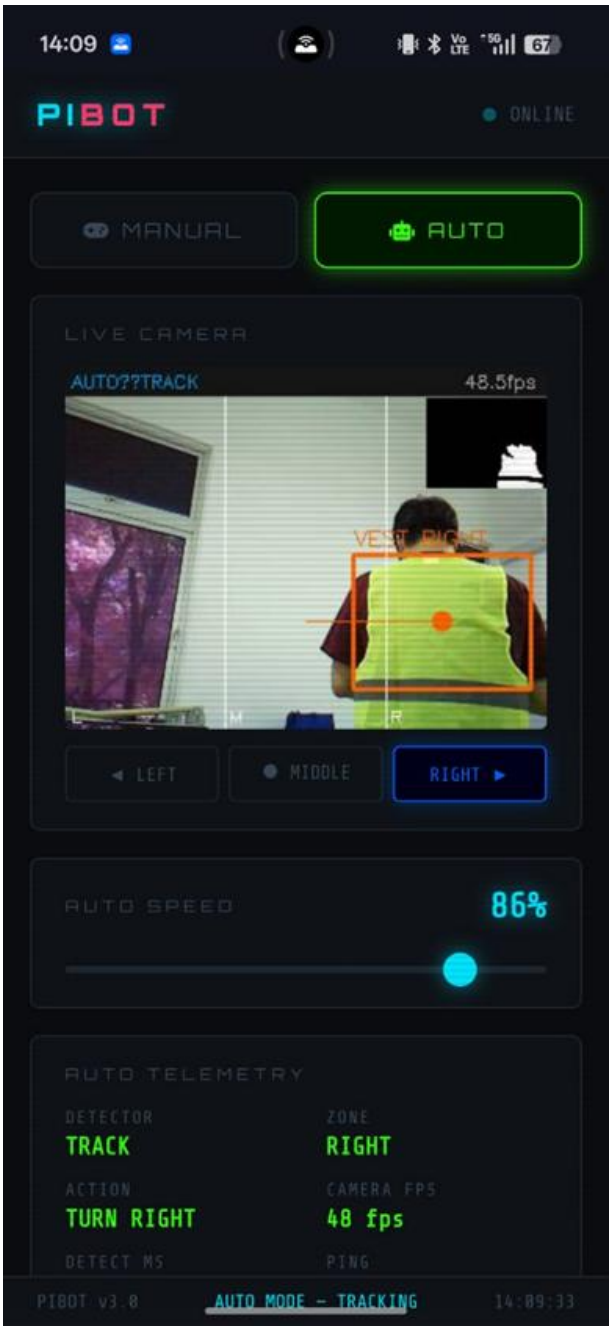


Figure 4.11: Web Interface Autofollowing Mode

4.6 Integrated System Tests

In the final integrated test, the complete robot, which is all threads running, battery powered, operating autonomously. The PiBot was evaluated to be over 10 test runs of 5 minutes each. In each run, a person wearing a hi-vis vest walked a predefined path in a laboratory room approximately 8 × 6 m in size. Cardboard box obstacles were placed at random positions in the room.

Table 4.5: Integrated System Performance Summary

Metric	Result
Target successfully followed for full 5-min run	9 out of 10 runs (90 %)
Obstacle collisions (physical contact with obstacle)	0 (zero)
Unintended stops (stopped when target visible, no obstacle)	1 (due to system crash)
Average following distance maintained	0.8 to 1.5 m
Full system crash or restart required	1

The most important result is zero obstacle collisions across all 10 integrated test runs and all 10 individual obstacle avoidance events that occurred during testing. This confirms that the IR priority override system works reliably in a real operating environment.

The 1 runs that failed to complete successfully were all due to the system crash, it may be caused by the overheating or the Raspberry Pi 3B+ had been operating for too long. This is a limitation of the Raspberry Pi 3B+ rather than the robot's software, and it could be addressed in future work by using a heatsink.

4.7 Summary

This chapter showed all the findings from the PiBot hardware and software testing, which used a Raspberry Pi 3B+ with a camera-based person-following system. Using a hybrid vision pipeline combining MobileNet SSD detection and colour tracking, the robot was able to follow a target wearing a hi-vis vest with a high success rate under real operating conditions. Upon acquiring the target, the system switched to lightweight colour tracking, which maintained 28 FPS while keeping CPU load at a manageable level. The BTS7960 motor driver handled peak currents of up to 25.6 A reliably, and the IR-based obstacle avoidance system recorded zero collisions across all integrated test runs. The system performed well in terms of responsiveness, detection accuracy, and physical control across all major components. These components include the vision pipeline, obstacle avoidance, motor control, and the web interface based on the performance evaluation. The results demonstrated that the PiBot achieves its design goals and provides a capable and practical autonomous load carrying platform suited for short to medium duration tasks in an indoor environment.

CHAPTER 5

CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusion

This project sets out to design and build PiBot, a low-cost autonomous mobile robot capable of following a person wearing a hi-vis safety vest around an indoor room environment, while keeping obstacle avoidance as the top priority at all times. Looking back at the four objectives that were set at the start, all of them were met by the end of the project.

The first objective was to build a strong and reliable chassis from a modified trolley frame that could handle heavy loads in a real workspace. This was achieved using two MY6812 24 V motors, BTS7960 motor driver modules, and a chain-and-gear transmission that provided enough torque to move significant weight. The frame held up well throughout all testing and proved capable of carrying heavy loads without any structural concerns.

The second objective was to develop a web dashboard that lets the user switch between manual steering and automatic following. This was delivered through a pure-Python WebSocket server paired with Flask MJPEG video streaming, giving the operator live camera footage alongside direct control buttons. The mean control latency came in at just 14 ms, and switching between modes worked reliably throughout every test.

The third objective was to implement a camera-based AI system that could accurately identify and follow workers in hi-vis vests. A two-stage pipeline was built for this purpose, which is MobileNet SSD, which handled the initial search and detection at 89% accuracy, and once the target was found, a colour tracker took over and maintained following at 28 FPS with far less processing demand. The handover between the two stages happened automatically without any input from the operator.

The fourth objective was to integrate infrared sensors with a priority-based algorithm to stop or steer the robot away from obstacles before any collision could happen. The symmetric debounce state machine fulfilled this requirement, recording zero obstacle collisions across all integrated test runs and eliminating the stuck-trigger fault entirely across 500 debounce trials.

Taken together, these results show that a safe, practical, and affordable person-following robot can be built entirely from off-the-shelf components and open-source software running on a Raspberry Pi 3B+. The design patterns developed here, particularly the priority-based motor command structure and the debounce algorithm, are straightforward enough to be carried across into other robotics projects where safety-critical responses need to work alongside both autonomous behaviour and manual control.

5.2 Recommendations

There are several improvements that stand out as worthwhile directions for future development based on what was learned during this project. The most impactful upgrade would be replacing the Raspberry Pi 3B+ with a processor with higher RAM, such as Raspberry Pi 5 or adding a Hailo-8 AI accelerator. Either option would allow the vision system to run YOLOv8 at 30+ FPS in real time, improving detection accuracy and pushing the reliable following range well beyond the current 3.5 m limit.

On the sensor side, swapping the binary IR sensors for VL53L1X time-of-flight sensors or radar sensors would give the robot actual distance readings rather than a simple present-or-absent signal. This would allow for smoother, more gradual obstacle avoidance instead of the hard turns that currently trigger at close range. Besides, adding wheel encoders would also help, giving the robot feedback on its actual speed and making it possible to maintain consistent movement regardless of how much weight it is carrying.

Then, person re-identification after tracking loss is another limitation worth tackling in future. Currently, losing sight of the target for more than roughly 1.2 seconds causes the system to restart its search entirely, which creates a risk of locking onto the wrong person in environments where several workers are present. Recording a colour signature of the vest when the target is first detected and using it to verify new detections after a loss, would make the system considerably more dependable in that kind of setting.

Next, a battery monitoring circuit would also be a sensible addition, as the current design has no way to detect when the pack is running low. This is because a low battery warning on the web interface and an automatic shutdown sequence would prevent accidental over-discharge and extend the battery's working life.

Finally, the chassis and wheels are only suited to flat indoor surfaces. Any move towards outdoor use on a construction site would need a more robust frame, larger tyres or tracks, and a camera better suited to handling bright outdoor lighting conditions.

5.3 Sustainable Development Goals (SDGs)

The PiBot project, whilst primarily an engineering undertaking, carries some meaningful connections to the United Nations Sustainable Development Goals that are worth acknowledging.

The strongest link sits with SDG 8, which is built around the idea of decent work and economic growth. There is a core reason for building PiBot in the first place to take the physical load off workers who spend entire shifts pushing and pulling those heavy trolleys through warehouses or across construction sites. That kind of repetitive manual work is a known source of injury, and a robot that can handle it reliably gives workers a chance to focus on less physically punishing tasks.

Then, this naturally overlaps with SDG 3, which deals with good health and wellbeing. For example, back injuries and musculoskeletal problems from manual handling are common in these environments and often have long term consequences for the people affected. Even a modest reduction in that kind of physical demand, achieved through affordable automation, is a step in the right direction.

Next, SDG 9 covers industry, innovation, and infrastructure, and PiBot speaks to that goal in a fairly direct way. The project was put together using components that anyone can buy off the shelf and software that is freely available, yet the end result is a robot that functions reliably in a real environment. Besides, that matters because it shows smaller organisations and research groups that practical automation is not out of reach just because they do not have access to large budgets or specialist manufacturing facilities.

And finally, there is also a quieter connection to SDG 4, which is about quality education. Thus, the project was carried out as part of an undergraduate degree, and everything from the hardware decisions to the software design was written up in detail enough that someone else could follow it, adapt it, or take it further. Sharing that kind of documented, hands-on work adds something small but worthwhile to the wider pool of accessible technical knowledge.

REFERENCES

Ahmad, S., Mehmood, F. and Kim, D.-H. (2019) 'A DIY approach for the design of mission-planning architecture using autonomous task–object mapping and the deployment model in mission-critical IoT systems', Sustainability, 11(13), p. 3647. Available at: https://doi.org/10.3390/su11133647. [Accessed 9 May 2026].

Akasiadis, C., Pitsilis, V. and Spyropoulos, C.D. (2019) 'A multi-protocol IoT platform based on open-source frameworks', Sensors, 19(19), p. 4217. Available at: https://doi.org/10.3390/s19194217. [Accessed 9 May 2026].

Bo, L., Xiao, Z. and Jingyi, Y. (2019). Design of Intelligent Shopping Cart for Supermarket. Journal of Physics: Conference Series, 1207, p.012005. doi:https://doi.org/10.1088/1742-6596/1207/1/012005. [Accessed 9 September 2025].

Espressif (2023). ESP32-WROOM-32 Datasheet. [online] Available at: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf. [Accessed 10 September 2025].

Handsontec. (n.d.). InfraRed IR Obstacle Detector. [online] Available at: https://www.handsontec.com/dataspecs/sensor/IR%20Obstacle%20Detector.pdf. [Accessed 10 September 2025].

Katona, K., Neamah, H.A. and Korondi, P. (2024) 'Obstacle avoidance and path planning methods for autonomous navigation of mobile robot', Sensors, 24(11), p. 3573. Available at: https://doi.org/10.3390/s24113573. [Accessed 9 May 2026].

Khade, N., Bambal, L., Wairagade, M., Bhardwaj, S. and Bhagat, S. (2024). Smart Trolley - Human Following Trolley. International Journal for Research in Applied Science and Engineering Technology, 12(4), pp.4968–4971. doi:https://doi.org/10.22214/ijraset.2024.60849. [Accessed 9 September 2025].

Le, N.-B.-V., Thai, H.-D., Yoon, C.-W. and Huh, J.-H. (2024) 'Recent development of drone technology software engineering: a systematic survey', IEEE Access, 12, pp. 128729–128751. Available at: https://doi.org/10.1109/access.2024.3454546. [Accessed 9 May 2026].

Leng Ng, Y., Siong Lim, C., A. Danapalasingam, K., Loong Peng Tan, M. and Wei Tan, C. (2015). Automatic Human Guided Shopping Trolley with Smart Shopping System. Jurnal Teknologi, 73(3). doi:https://doi.org/10.11113/jt.v73.4246. [Accessed 9 September 2025].

Luthuli, M.B., Malele, V. and Owolawi, P.A. (2022) 'Smart walk: a smart stick for the visually impaired', in International Conference on Intelligent and Innovative Computing Applications (ICONIC). Plaine Magnien, Mauritius, pp. 131–136. Available at: https://doi.org/10.59200/iconic.2022.014. [Accessed 9 May 2026].

Pehlivan, S., Unay, M. and Akan, A. (2019) 'Designing an obstacle detection and alerting system for visually impaired people on sidewalks', in 2019 Medical Technologies Congress (TIPTEKNO). Izmir, Turkey: IEEE, pp. 1–4. Available at: https://doi.org/10.1109/tiptekno.2019.8895181. [Accessed 9 May 2026].

Raspberry Pi (n.d.). Raspberry Pi Documentation - Camera. [online] Raspberry pi. Available at: https://www.raspberrypi.com/documentation/accessories/camera.html. [Accessed 9 September 2025].

Raspberry Pi (n.d.). Raspberry Pi Documentation - Camera. [online] Raspberry pi. Available at: https://www.raspberrypi.com/documentation/accessories/camera.html. [Accessed 9 September 2025].

Raspberry Pi 3 Model B+. (2023). Raspberry Pi 3 Model B+. Available at: https://datasheets.raspberrypi.com/rpi3/raspberry-pi-3-b-plus-product-brief.pdf. [Accessed 10 September 2025].

Reis, D., Kupec, J., Hong, J. and Daoudi, A. (2023). Real-Time Flying Object Detection with YOLOv8. doi:https://doi.org/10.48550/arxiv.2305.09972. [Accessed 10 September 2025].

Tang, X., Zhang, Z. and Qin, Y. (2022). On-Road Object Detection and Tracking Based on Radar and Vision Fusion: A Review. 14(5), pp.103–128. doi:https://doi.org/10.1109/mits.2021.3093379. [Accessed 10 September 2025].

Ultralytics. (n.d.). Ultralytics | Revolutionizing the World of Vision AI. [online] Available at: https://www.ultralytics.com/. [Accessed 11 September 2025].

Weltner, D.T. (2025). ESP32 DevKitC V4 - DONE.LAND. [online] Done.land. Available at: https://done.land/components/microcontroller/families/esp/esp32/developmentboards/esp32s/esp32devkitcv4/ [Accessed 11 Sep. 2025].

Xu, Z., Zhan, X., Yumeng Xiu, Suzuki, C. and Shimada, K. (2024). Onboard Dynamic-Object Detection and Tracking for Autonomous Robot Navigation With RGB-D Camera. IEEE Robotics and Automation Letters, 9(1), pp.651–658. doi:https://doi.org/10.1109/lra.2023.3334683. [Accessed 10 September 2025].

Yalumotor. (2019). 12V/24V Electric Scooter EBike mini DC Motor MY6812 100W 120W 150W - China Yalu Electric Yalumotor. [online] Available at: https://www.yalumotor.com/high-speed-brush-dc-motor/12v-24v-electric-scooter-ebike-mini-dc-motor-my6812-100w-120w-150w [Accessed 12 Sep. 2025].

APPENDIX

Appendix A: PiBot Python Code

```
from flask import Flask, render_template, Response
import RPi.GPIO as GPIO
import atexit, socket, struct, hashlib, base64, cv2, numpy as np, os, time, threading
from picamera2 import Picamera2

app = Flask(__name__)

# ═══════════════════════════════════════════════════════════════════════════
# ═══════════════════════════
#  GPIO / MOTOR PINS
# ═══════════════════════════════════════════════════════════════════════════
# ═══════════════════════════


# Left motor (BTS7960)
L_RPWM, L_LPWM, L_L_EN, L_R_EN = 5, 6, 13, 16

# Right motor (BTS7960)
R_RPWM, R_LPWM, R_L_EN, R_R_EN = 19, 26, 20, 21

# IR sensors (MH-B: LOW = detected/black line, HIGH = clear)
IR_LEFT  = 2
IR_MID   = 4   # middle sensor — stop on any detection involving mid
IR_RIGHT = 3

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

for _p in [L_RPWM, L_LPWM, L_L_EN, L_R_EN, R_RPWM, R_LPWM, R_L_EN, R_R_EN]:
    GPIO.setup(_p, GPIO.OUT)
```

```
for _e in [L_L_EN, L_R_EN, R_L_EN, R_R_EN]:
    GPIO.output(_e, True)

GPIO.setup(IR_LEFT,  GPIO.IN)
GPIO.setup(IR_MID,   GPIO.IN)
GPIO.setup(IR_RIGHT, GPIO.IN)

l_rpwm = GPIO.PWM(L_RPWM, 100); l_lpwm = GPIO.PWM(L_LPWM, 100)
r_rpwm = GPIO.PWM(R_RPWM, 100); r_lpwm = GPIO.PWM(R_LPWM, 100)
for _p in [l_rpwm, l_lpwm, r_rpwm, r_lpwm]: _p.start(0)

SPEED      = 20
MODE       = 'manual'
motor_lock = threading.Lock()

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════
#  IR OVERRIDE STATE
#  ir_override is True whenever an IR sensor is actively detecting.
#  While True, ALL motor commands from manual/auto are silently blocked.
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════

ir_override = False   # set by IR thread; read by motor helpers + detection loop

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════
#  MOTOR PRIMITIVES  (all respect ir_override)
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════

def _raw_stop():
    """Unconditional stop — used internally and by IR thread."""
    for p in [l_rpwm, l_lpwm, r_rpwm, r_lpwm]: p.ChangeDutyCycle(0)

def _raw_forward():
    _raw_stop();                                    l_lpwm.ChangeDutyCycle(SPEED);
r_rpwm.ChangeDutyCycle(SPEED)

def _raw_backward():
    _raw_stop();                                    l_rpwm.ChangeDutyCycle(SPEED);
```

```
r_lpwm.ChangeDutyCycle(SPEED)

def _raw_turn_left():
    """Right motor forward only — pivot left."""
    _raw_stop(); r_rpwm.ChangeDutyCycle(SPEED)

def _raw_turn_right():
    """Left motor forward only — pivot right."""
    _raw_stop(); l_lpwm.ChangeDutyCycle(SPEED)

# Public wrappers used by manual/auto — silently no-op when IR is active
def stop():
    if ir_override: return
    with motor_lock: _raw_stop()
    print(f"[MOTOR] STOP")

def forward():
    if ir_override: return
    with motor_lock: _raw_forward()
    print(f"[MOTOR] FWD {SPEED}%")

def backward():
    if ir_override: return
    with motor_lock: _raw_backward()
    print(f"[MOTOR] BWD {SPEED}%")

def turn_left():
    if ir_override: return
    with motor_lock: _raw_turn_left()
    print(f"[MOTOR] LEFT {SPEED}%")

def turn_right():
    if ir_override: return
    with motor_lock: _raw_turn_right()
    print(f"[MOTOR] RIGHT {SPEED}%")

DRIVE_MAP = {'w': forward, 'x': backward, 'a': turn_left, 'd': turn_right, 's': stop}

#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════
#  IR SENSOR THREAD  (highest priority — preempts manual and auto)
#
#  Sensor logic (MH-B module: LOW = line/obstacle detected):
#    Both detected → STOP
```

```
#   Left only     → TURN RIGHT  (steer away from left obstacle)
#   Right only    → TURN LEFT   (steer away from right obstacle)
#   Neither       → release override, let manual/auto resume
#
#  Broadcasts "IR:<state>" to all WS clients so the UI can show it.
#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════

IR_POLL_HZ       = 50    # sample rate Hz — faster sampling improves debounce
resolution
IR_DEBOUNCE_N   = 5    # consecutive identical reads needed to confirm any state
change
#
# Root cause of stuck-trigger:
#   A single CLEAR read was enough to release the override, but a single glitch
#   was also enough to SET it. This asymmetry means any momentary noise latches
#   the override on permanently. Fix: BOTH trigger and clear require N consecutive
#   matching reads before the confirmed state changes.

def _raw_read_sensors():
    """Read all three IR sensors. Returns (left, mid, right) bools. True = detected."""
    return (
        not GPIO.input(IR_LEFT),
        not GPIO.input(IR_MID),
        not GPIO.input(IR_RIGHT),
    )

def _classify(left_det, mid_det, right_det):
    """Map sensor booleans to a state string using the trigger truth table."""
    if not (left_det or mid_det or right_det):
        return "CLEAR"
    if mid_det:
        return "STOP"
    if left_det and right_det:
        return "STOP"
    if left_det:
        return "TURN_RIGHT"
    return "TURN_LEFT"

def _apply_state(state):
    """Drive motors according to confirmed IR state. Called inside motor_lock."""
    global ir_override
    if state == "CLEAR":
        ir_override = False
```

```
        _raw_stop()   # safe handoff — manual/auto will re-command on next cycle
    else:
        ir_override = True
        if state == "STOP":
            _raw_stop()
        elif state == "TURN_RIGHT":
            _raw_stop()
            l_lpwm.ChangeDutyCycle(SPEED)   # left motor forward -> pivot right
        elif state == "TURN_LEFT":
            _raw_stop()
            r_rpwm.ChangeDutyCycle(SPEED)   # right motor forward -> pivot left

def ir_sensor_loop():
    """
    Debounced IR sensor loop.

    A candidate state must be read N consecutive times before it becomes the
    confirmed state. This prevents a single glitch — in either direction — from
    latching or releasing the override incorrectly.

    Truth table (MH-B: LOW = detected):
      mid (any combo)    -> STOP
      left + right       -> STOP
      left only          -> TURN_RIGHT
      right only         -> TURN_LEFT
      none               -> CLEAR
    """
    global ir_override
    confirmed_state  = "CLEAR"   # last state that passed debounce
    candidate_state  = "CLEAR"   # state we are currently trying to confirm
    candidate_count  = 0         # how many consecutive reads matched candidate
    prev_broadcast   = None

    while True:
        left_det, mid_det, right_det = _raw_read_sensors()
        raw_state = _classify(left_det, mid_det, right_det)

        if raw_state == candidate_state:
            candidate_count += 1
        else:
            # New candidate — reset counter
            candidate_state = raw_state
            candidate_count = 1

        if candidate_count >= IR_DEBOUNCE_N and candidate_state != confirmed_state:
```

```
            # Debounce passed — apply the new confirmed state
            confirmed_state = candidate_state
            with motor_lock:
                _apply_state(confirmed_state)
            print(f"[IR] confirmed={confirmed_state}  "
                  f"left={left_det}  mid={mid_det}  right={right_det}")

        if confirmed_state != prev_broadcast:
            _broadcast(f"IR:{confirmed_state}")
            prev_broadcast = confirmed_state

        time.sleep(1.0 / IR_POLL_HZ)

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════
#  WEBSOCKET CLIENT REGISTRY
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════

_ws_clients      = set()
_ws_clients_lock = threading.Lock()

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════
#  DETECTION CONFIG
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════

FRAME_W = 320; FRAME_H = 240
CONFIDENCE_MIN = 0.4; TARGET_CLASS = 15; DETECT_EVERY = 3

# Hi-vis vest colour filter (yellow AND green vests)
VEST_G_MIN      = 80
VEST_B_MAX      = 160
VEST_GB_DIFF    = 25
VEST_PIX_MIN    = 60
MIN_CONTOUR_AREA = 400
COLOR_MISS_LIMIT = 35

one_third  = FRAME_W // 3
two_thirds = FRAME_W * 2 // 3
```

```
C_LEFT   = (255, 100, 0); C_MIDDLE = (0, 200, 0); C_RIGHT = (0, 100, 255)
C_SEARCH = (255, 165, 0); C_WHITE  = (255, 255, 255)
C_FPS    = (180, 180, 180); C_DARK  = (20, 20, 20)

_base = os.path.dirname(os.path.abspath(__file__))
net = None; NET_AVAILABLE = False
_proto = os.path.join(_base, 'deploy.prototxt')
_caffe = os.path.join(_base, 'mobilenet.caffemodel')
if os.path.exists(_proto) and os.path.exists(_caffe):
    try:
        net = cv2.dnn.readNetFromCaffe(_proto, _caffe)
        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
        NET_AVAILABLE = True; print("[Model] MobileNet OK")
    except cv2.error as e:
        print(f"[Model] {e}")

_latest_jpeg = None; _jpeg_lock = threading.Lock()

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════
#  CAMERA THREAD
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════


class CameraThread:
    def __init__(self, picam2):
        self.picam2 = picam2; self.frame = None
        self.lock = threading.Lock(); self.running = True
        threading.Thread(target=self._loop, daemon=True).start()

    def _loop(self):
        while self.running:
            f = self.picam2.capture_array()
            with self.lock: self.frame = f

    def read(self):
        with self.lock: return self.frame.copy() if self.frame is not None else None

    def stop(self): self.running = False

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════════
```

```
# DETECTION HELPERS
#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════


def _vest_roi(roi):
    if roi.size == 0: return False
    mean_v = float(np.mean(roi))
    if mean_v < 40: return False
    if mean_v < 80:
        scale = 80.0 / max(mean_v, 1)
        roi = np.clip(roi.astype(np.float32) * scale, 0, 255).astype(np.uint8)
    r = roi[:, :, 0].astype(np.float32)
    g = roi[:, :, 1].astype(np.float32)
    b = roi[:, :, 2].astype(np.float32)
    sat  = g - (r + b) / 2.0
    mask = (g > VEST_G_MIN) & (b < VEST_B_MAX) & (g > b + VEST_GB_DIFF)
& (g > r * 0.6) & (sat > 30)
    return int(np.count_nonzero(mask)) > VEST_PIX_MIN

def mobilenet_detect(rgb):
    if not NET_AVAILABLE: return None
    bgr = cv2.cvtColor(rgb, cv2.COLOR_RGB2BGR); h, w = rgb.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(bgr, (300, 300)), 0.007843, (300, 300),
127.5)
    net.setInput(blob); dets = net.forward()
    best, ba = None, 0
    for i in range(dets.shape[2]):
        conf = float(dets[0, 0, i, 2])
        if conf < CONFIDENCE_MIN or int(dets[0, 0, i, 1]) != TARGET_CLASS:
continue
        box = (dets[0, 0, i, 3:7] * np.array([w, h, w, h])).astype(int)
        x1, y1, x2, y2 = max(0, box[0]), max(0, box[1]), min(w, box[2]), min(h, box[3])
        if not _vest_roi(rgb[y1:y2, x1:x2]): continue
        area = (x2 - x1) * (y2 - y1)
        if area > ba: ba = area; best = ((x1 + x2) // 2, (y1 + y2) // 2, x1, y1, x2, y2)
    return best

def color_track(rgb):
    """Track hi-vis vest — rejects white ceiling, blue objects, grey walls."""
    lab = cv2.cvtColor(rgb, cv2.COLOR_RGB2LAB)
    l, a, b_ch = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(4, 4))
    l = clahe.apply(l)
    rgb_eq = cv2.cvtColor(cv2.merge([l, a, b_ch]), cv2.COLOR_LAB2RGB)
```

```
    hsv = cv2.cvtColor(rgb_eq, cv2.COLOR_RGB2HSV)
    mask_hsv = cv2.inRange(hsv,
                           np.array([38, 120, 60]),
                           np.array([82, 255, 255]))

    r = rgb_eq[:, :, 0].astype(np.float32)
    g = rgb_eq[:, :, 1].astype(np.float32)
    b = rgb_eq[:, :, 2].astype(np.float32)
    sat    = g - (r + b) / 2.0
    mask_rgb = (
        (g > VEST_G_MIN) & (b < VEST_B_MAX) & (g > b + VEST_GB_DIFF) &
        (g > r * 0.6) & (sat > 30)
    ).astype(np.uint8) * 255

    mask = cv2.bitwise_or(mask_rgb, mask_hsv)
    k    = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    mask = cv2.erode(mask, k, iterations=1)
    mask = cv2.dilate(mask, k, iterations=2)

    cnts,          _          =          cv2.findContours(mask,          cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    if not cnts: return None, mask
    lg   = max(cnts, key=cv2.contourArea)
    area = cv2.contourArea(lg)
    if area < MIN_CONTOUR_AREA: return None, mask
    x, y, bw, bh = cv2.boundingRect(lg)
    return (x + bw // 2, y + bh // 2, x, y, x + bw, y + bh, area), mask

#
═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════
#  DETECTION + STREAM LOOP
#
═══════════════════════════════════════════════════════════════════════════════
═══════════════════════════════════════


def get_zone(cx):
    if cx < one_third:  return "LEFT",   C_LEFT
    if cx < two_thirds: return "MIDDLE", C_MIDDLE
    return "RIGHT", C_RIGHT

def detection_loop(cam):
    global _latest_jpeg
    det_mode   = "search" if NET_AVAILABLE else "track"
    color_miss = 0; frame_num = 0; fps_times = []; detect_ms = 0.0
```

```
    while True:
        rgb = cam.read()
        if rgb is None: time.sleep(0.01); continue

        frame_num += 1
        now = time.time()
        fps_times = [t for t in fps_times if now - t < 2.0]; fps_times.append(now)
        fps = len(fps_times) / 2.0

        result     = None
        color_mask = np.zeros((FRAME_H, FRAME_W), dtype=np.uint8)

        if MODE == 'auto':
            if det_mode == "search":
                if frame_num % DETECT_EVERY == 0:
                    t0 = time.time(); detected = mobilenet_detect(rgb); detect_ms =
(time.time() - t0) * 1000
                    if detected: det_mode = "track"; color_miss = 0; result = detected
            elif det_mode == "track":
                tracked, color_mask = color_track(rgb)
                if tracked:
                    color_miss = 0; cx, cy, x1, y1, x2, y2, _ = tracked; result = (cx, cy, x1,
y1, x2, y2)
                else:
                    color_miss += 1
                    if color_miss >= COLOR_MISS_LIMIT:
                        det_mode = "search" if NET_AVAILABLE else "track"; color_miss =
0

            # Motor commands only issued when IR is NOT overriding
            if not ir_override:
                if result:
                    zone, _ = get_zone(result[0])
                    if zone == "LEFT":   turn_left()
                    elif zone == "RIGHT": turn_right()
                    else:                 forward()
                else:
                    stop()

            zone_name = get_zone(result[0])[0] if result else "NONE"
            _broadcast(f"AUTO:{zone_name}:{det_mode}:{round(fps,
1)}:{round(detect_ms)}")

        #                ──                Build                JPEG                overlay
──────────────────────────────────────────────────────────────────────────────────────────

        display = rgb.copy()
```

```
        # IR indicator banner (shown whenever override is active)
        if ir_override:
            cv2.rectangle(display, (0, FRAME_H - 20), (FRAME_W, FRAME_H), (200,
0, 0), -1)
            cv2.putText(display, "IR OVERRIDE", (4, FRAME_H - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 255, 255), 1)

        if result:
            cx, cy, x1, y1, x2, y2 = result; zone, col = get_zone(cx)
            cv2.rectangle(display, (x1, y1), (x2, y2), col, 2)
            cv2.circle(display, (cx, cy), 7, col, -1)
            cv2.putText(display, f"VEST {zone}", (x1, max(y1 - 6, 10)),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, col, 1)

        cv2.line(display, (one_third, 0),  (one_third, FRAME_H),  C_WHITE, 1)
        cv2.line(display, (two_thirds, 0), (two_thirds, FRAME_H), C_WHITE, 1)
        cv2.putText(display,          "L",          (4,          FRAME_H          -          5),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, C_WHITE, 1)
        cv2.putText(display,    "M",    (one_third    +    3,    FRAME_H    -    5),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, C_WHITE, 1)
        cv2.putText(display,    "R",    (two_thirds    +    3,    FRAME_H    -
5),cv2.FONT_HERSHEY_SIMPLEX, 0.35, C_WHITE, 1)

        badge     = "AUTO-" + det_mode.upper() if MODE == 'auto' else "MANUAL"
        badge_col = C_SEARCH if MODE == 'auto' else C_MIDDLE
        cv2.rectangle(display, (0, 0), (FRAME_W, 18), C_DARK, -1)
        cv2.putText(display,     badge,                                          (4,     13),
cv2.FONT_HERSHEY_SIMPLEX, 0.4, badge_col, 1)
        cv2.putText(display,          f"{fps:.1f}fps",(FRAME_W          -          55,
13),cv2.FONT_HERSHEY_SIMPLEX, 0.4, C_FPS, 1)

        if MODE == 'auto':
            thumb = cv2.resize(color_mask, (80, 60))
            display[20:80,  FRAME_W  -  80:FRAME_W]  =  cv2.cvtColor(thumb,
cv2.COLOR_GRAY2RGB)

        ok, jpg = cv2.imencode('.jpg', display, [cv2.IMWRITE_JPEG_QUALITY, 70])
        if ok:
            with _jpeg_lock: _latest_jpeg = jpg.tobytes()

#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════
#  FLASK ROUTES
#
```

```
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════

@app.route('/')
def index(): return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    def generate():
        while True:
            with _jpeg_lock: frame = _latest_jpeg
            if frame:
                yield b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + frame + b'\r\n'
            time.sleep(0.033)
    return Response(generate(), mimetype='multipart/x-mixed-replace;
boundary=frame')

#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════
#  PURE STDLIB WEBSOCKET SERVER  (port 5001)
#
═══════════════════════════════════════════════════════════════════════
═══════════════════════════════════════

def _ws_handshake(conn):
    data = b''
    while b'\r\n\r\n' not in data:
        chunk = conn.recv(1024)
        if not chunk: return False
        data += chunk
    headers = {}
    for line in data.decode('utf-8', errors='ignore').split('\r\n')[1:]:
        if ':' in line:
            k, v = line.split(':', 1)
            headers[k.strip().lower()] = v.strip()
    key    = headers.get('sec-websocket-key', '')
    accept = base64.b64encode(
        hashlib.sha1((key                    +                    '258EAFA5-E914-47DA-95CA-
C5AB0DC85B11').encode()).digest()
    ).decode()
    response = (
        'HTTP/1.1 101 Switching Protocols\r\n'
        'Upgrade: websocket\r\n'
        'Connection: Upgrade\r\n'
        f'Sec-WebSocket-Accept: {accept}\r\n\r\n'
```

```
    )
    conn.sendall(response.encode())
    return True

def _ws_recv(conn):
    try:
        header = b''
        while len(header) < 2:
            b = conn.recv(2 - len(header))
            if not b: return None
            header += b
        fin_op     = header[0]; masked_len = header[1]
        opcode     = fin_op & 0x0f
        if opcode == 0x8: return None
        length     = masked_len & 0x7f
        if length == 126:  length = struct.unpack('>H', conn.recv(2))[0]
        elif length == 127: length = struct.unpack('>Q', conn.recv(8))[0]
        mask    = conn.recv(4) if (masked_len & 0x80) else None
        payload = b''
        while len(payload) < length:
            chunk = conn.recv(length - len(payload))
            if not chunk: return None
            payload += chunk
        if mask:
            payload = bytes(b ^ mask[i % 4] for i, b in enumerate(payload))
        return payload.decode('utf-8', errors='ignore')
    except Exception:
        return None

def _ws_send(conn, text):
    try:
        payload = text.encode('utf-8')
        length  = len(payload)
        if length < 126:
            header = bytes([0x81, length])
        elif length < 65536:
            header = bytes([0x81, 126]) + struct.pack('>H', length)
        else:
            header = bytes([0x81, 127]) + struct.pack('>Q', length)
        conn.sendall(header + payload)
        return True
    except Exception:
        return False

def _handle_ws_client(conn, addr):
    global MODE, SPEED
```

```
    print(f"[WS] {addr} connected")
    if not _ws_handshake(conn):
        conn.close(); return
    with _ws_clients_lock: _ws_clients.add(conn)
    _ws_send(conn, f"HELLO:mode={MODE}:net={NET_AVAILABLE}")
    try:
        while True:
            msg = _ws_recv(conn)
            if msg is None: break
            msg = msg.strip()
            if msg == 'PING':
                _ws_send(conn, 'PONG')
            elif msg.startswith('DRIVE:') and MODE == 'manual':
                # IR override blocks manual drive commands silently
                DRIVE_MAP.get(msg.split(':')[1], stop)()
            elif msg.startswith('SPEED:'):
                SPEED = max(10, min(100, int(msg.split(':')[1])))
                print(f"[WS] speed={SPEED}")
            elif msg.startswith('MODE:'):
                MODE = msg.split(':')[1]
                print(f"[WS] mode={MODE}")
                if MODE == 'manual': stop()
                _broadcast(f"MODE:{MODE}")
    except Exception as e:
        print(f"[WS] {addr} error: {e}")
    finally:
        with _ws_clients_lock: _ws_clients.discard(conn)
        stop()
        conn.close()
        print(f"[WS] {addr} disconnected")

def _broadcast(msg):
    with _ws_clients_lock:
        dead = set()
        for conn in _ws_clients:
            if not _ws_send(conn, msg): dead.add(conn)
        if dead: _ws_clients.difference_update(dead)

def start_ws_server(port=5001):
    srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    srv.bind(('0.0.0.0', port)); srv.listen(10)
    print(f"[WS] Listening on :{port}")
    while True:
        conn, addr = srv.accept()
        threading.Thread(target=_handle_ws_client,          args=(conn,          addr),
```

```
daemon=True).start()

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════
#  CLEANUP
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════

@atexit.register
def cleanup():
    _raw_stop()
    GPIO.cleanup()

# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════
#  MAIN
# ═══════════════════════════════════════════════════════════════════════
# ═══════════════════════════════════════

if __name__ == '__main__':
    picam2 = Picamera2()
    picam2.configure(picam2.create_video_configuration(
        main={"format": "RGB888", "size": (FRAME_W, FRAME_H)},
        controls={"FrameDurationLimits": (33333, 33333)}
    ))
    picam2.start(); time.sleep(2.0)

    cam = CameraThread(picam2)
    threading.Thread(target=detection_loop, args=(cam,), daemon=True).start()

    # IR sensor thread — runs at highest priority, no dependency on MODE
    threading.Thread(target=ir_sensor_loop, daemon=True).start()

    # Raw WebSocket server on port 5001
    threading.Thread(target=start_ws_server, args=(5001,), daemon=True).start()

    print("[PiBot] HTTP  → http://0.0.0.0:5000")
    print("[PiBot] WS    → ws://0.0.0.0:5001")
    app.run(host='0.0.0.0', port=5000, debug=False,
            use_reloader=False, threaded=True)
```

Appendix B: Web Interface Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
  <title>Pi Bot Controller</title>
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link href="https://fonts.googleapis.com/css2?family=Share+Tech+Mono&family=Orbitron:wght@400;700;900&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css">
  <style>
    :root {
      --bg:      #0a0c0f;
      --panel:   #0f1318;
      --border:  #1e2830;
      --accent:  #00e5ff;
      --accent2: #ff3d71;
      --accent3: #39ff14;  /* auto-mode green */
      --text:    #c8d8e8;
      --dim:     #3a4a5a;
      --glow:    0 0 12px #00e5ff88;
      --glow-red: 0 0 12px #ff3d7188;
      --glow-grn: 0 0 12px #39ff1488;
    }

    * { box-sizing: border-box; margin: 0; padding: 0; }

    body {
      font-family: 'Share Tech Mono', monospace;
      background: var(--bg);
      color: var(--text);
      min-height: 100vh;
      display: flex;
      flex-direction: column;
      align-items: center;
    }

    /* Scanline overlay */
    body::before {
      content: '';
```

```
    position: fixed;
    inset: 0;
    background: repeating-linear-gradient(
      0deg, transparent, transparent 2px,
      rgba(0,0,0,0.07) 2px, rgba(0,0,0,0.07) 4px
    );
    pointer-events: none;
    z-index: 999;
  }

  /* ── Header ── */
  header {
    width: 100%;
    padding: 14px 20px;
    display: flex;
    align-items: center;
    justify-content: space-between;
    border-bottom: 1px solid var(--border);
    background: var(--panel);
  }

  .logo {
    font-family: 'Orbitron', sans-serif;
    font-weight: 900;
    font-size: 1.1rem;
    letter-spacing: 4px;
    color: var(--accent);
    text-shadow: var(--glow);
  }
  .logo span { color: var(--accent2); }

  .header-right {
    display: flex;
    align-items: center;
    gap: 8px;
    font-size: 0.75rem;
    color: var(--dim);
  }

  .dot {
    width: 8px; height: 8px;
    border-radius: 50%;
    background: var(--accent2);
    animation: pulse 1.5s infinite;
  }
  .dot.connected { background: var(--accent); }
```

```
@keyframes pulse {
  0%, 100% { opacity: 1; }
  50%      { opacity: 0.3; }
}

/* ── Main ── */
main {
  width: 100%;
  max-width: 540px;
  padding: 20px 16px 80px;
  display: flex;
  flex-direction: column;
  gap: 16px;
}

/* ── Mode Toggle ── */
.mode-toggle {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 10px;
}

.mode-btn {
  font-family: 'Orbitron', sans-serif;
  font-size: 0.75rem;
  letter-spacing: 2px;
  padding: 14px 8px;
  border: 1px solid var(--border);
  border-radius: 8px;
  background: var(--panel);
  color: var(--dim);
  cursor: pointer;
  transition: all 0.15s;
  text-align: center;
  user-select: none;
}

.mode-btn i { margin-right: 6px; }

.mode-btn.active-manual {
  color: var(--accent);
  border-color: var(--accent);
  box-shadow: var(--glow);
  background: #001f2a;
}
```

```
.mode-btn.active-auto {
  color: var(--accent3);
  border-color: var(--accent3);
  box-shadow: var(--glow-grn);
  background: #001a00;
}

/* ── Section card ── */
.card {
  border: 1px solid var(--border);
  border-radius: 6px;
  padding: 16px 20px;
  background: var(--panel);
}

.section-title {
  font-family: 'Orbitron', sans-serif;
  font-size: 0.6rem;
  letter-spacing: 3px;
  color: var(--dim);
  margin-bottom: 14px;
}

/* ── Camera feed ── */
.cam-wrap {
  position: relative;
  width: 100%;
  aspect-ratio: 4/3;
  background: #000;
  border-radius: 4px;
  overflow: hidden;
}

.cam-wrap img {
  width: 100%;
  height: 100%;
  object-fit: cover;
  display: block;
}

.cam-overlay {
  position: absolute;
  inset: 0;
  display: flex;
  align-items: center;
```

```
        justify-content: center;
        color: var(--dim);
        font-size: 0.75rem;
        pointer-events: none;
      }

      /* Zone indicator bar */
      .zone-bar {
        display: grid;
        grid-template-columns: 1fr 1fr 1fr;
        gap: 6px;
        margin-top: 10px;
      }

      .zone-cell {
        text-align: center;
        padding: 8px 4px;
        border: 1px solid var(--border);
        border-radius: 4px;
        font-size: 0.7rem;
        color: var(--dim);
        transition: all 0.15s;
      }

      .zone-cell.active-left  { color: #ff6400; border-color: #ff6400; background: #1a0a00; box-shadow: 0 0 10px #ff640066; }
      .zone-cell.active-mid   { color: var(--accent3); border-color: var(--accent3); background: #001a00; box-shadow: var(--glow-grn); }
      .zone-cell.active-right { color: #0064ff; border-color: #0064ff; background: #00001a; box-shadow: 0 0 10px #0064ff66; }

      /* ── Speed Slider ── */
      .speed-header {
        display: flex;
        justify-content: space-between;
        align-items: center;
        margin-bottom: 14px;
      }

      .speed-val {
        font-size: 1.3rem;
        font-weight: bold;
        color: var(--accent);
        text-shadow: var(--glow);
      }
```

```
input[type=range] {
  -webkit-appearance: none;
  width: 100%;
  height: 4px;
  background: var(--border);
  border-radius: 2px;
  outline: none;
}
input[type=range]::-webkit-slider-thumb {
  -webkit-appearance: none;
  width: 20px; height: 20px;
  border-radius: 50%;
  background: var(--accent);
  box-shadow: var(--glow);
  cursor: pointer;
}

/* ── D-Pad ── */
.dpad-wrap {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 16px;
}

.dpad {
  display: grid;
  grid-template-columns: repeat(3, 80px);
  grid-template-rows: repeat(3, 80px);
  gap: 8px;
}

.btn {
  display: flex;
  align-items: center;
  justify-content: center;
  width: 80px; height: 80px;
  font-size: 1.4rem;
  background: var(--panel);
  color: var(--dim);
  border: 1px solid var(--border);
  border-radius: 8px;
  cursor: pointer;
  transition: all 0.08s;
  user-select: none;
  -webkit-tap-highlight-color: transparent;
```

```
}

.btn:active, .btn.active {
  color: var(--accent);
  border-color: var(--accent);
  box-shadow: var(--glow), inset 0 0 20px #00e5ff18;
  transform: scale(0.93);
}

.btn-stop {
  background: #1a0a0a;
  border-color: #3a1515;
  color: #6a2525;
}
.btn-stop:active, .btn-stop.active {
  color: var(--accent2);
  border-color: var(--accent2);
  box-shadow: var(--glow-red), inset 0 0 20px #ff3d7118;
}

.btn-empty { visibility: hidden; }

/* ── Telemetry ── */
.telemetry {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 12px 24px;
}

.tele-item { display: flex; flex-direction: column; gap: 3px; }
.tele-label { font-size: 0.65rem; color: var(--dim); letter-spacing: 1px; }
.tele-val   { font-size: 0.9rem; color: var(--accent); font-weight: bold; }
.tele-val.grn { color: var(--accent3); }
.tele-val.red { color: var(--accent2); }

/* ── Hidden panel logic ── */
.panel-manual, .panel-auto { display: none; }
body.mode-manual .panel-manual { display: contents; }
body.mode-auto   .panel-auto   { display: contents; }
/* camera always visible */
.panel-cam { display: block; }

/* ── Status Bar ── */
.statusbar {
  position: fixed;
  bottom: 0; left: 0; right: 0;
```

```
        padding: 8px 20px;
        border-top: 1px solid var(--border);
        background: var(--panel);
        display: flex;
        justify-content: space-between;
        font-size: 0.7rem;
        color: var(--dim);
        z-index: 100;
      }
      #status-text { color: var(--accent); }
    </style>
</head>
<body class="mode-manual">

<header>
  <div class="logo">PI<span>BOT</span></div>
  <div class="header-right">
    <div class="dot" id="conn-dot"></div>
    <span id="conn-label">OFFLINE</span>
  </div>
</header>

<main>

  <!-- ── Mode Toggle ── -->
  <div class="mode-toggle">
    <div class="mode-btn active-manual" id="btn-mode-manual"
onclick="setMode('manual')">
      <i class="fas fa-gamepad"></i>MANUAL
    </div>
    <div class="mode-btn" id="btn-mode-auto" onclick="setMode('auto')">
      <i class="fas fa-robot"></i>AUTO
    </div>
  </div>

  <!-- ── Camera Feed (always visible) ── -->
  <div class="card panel-cam">
    <div class="section-title">LIVE CAMERA</div>
    <div class="cam-wrap">
      <img id="cam-feed" src="/video_feed" alt="camera">
      <div class="cam-overlay" id="cam-overlay" style="display:none">NO
SIGNAL</div>
    </div>

    <!-- Auto-mode zone indicator (shown only in auto) -->
    <div class="zone-bar" id="zone-bar" style="display:none; margin-top:10px;">
```

```
      <div class="zone-cell" id="zone-left">◀ LEFT</div>
      <div class="zone-cell" id="zone-mid">● MIDDLE</div>
      <div class="zone-cell" id="zone-right">RIGHT ▶</div>
    </div>
  </div>

  <!-- ═══ MANUAL PANEL ═══ -->
  <div class="panel-manual">

    <!-- Speed -->
    <div class="card">
      <div class="speed-header">
        <div class="section-title" style="margin:0">MOTOR SPEED</div>
        <div class="speed-val"><span id="speed-display">20</span>%</div>
      </div>
      <input type="range" min="10" max="100" value="20" id="speed-slider">
    </div>

    <!-- D-Pad -->
    <div class="card">
      <div class="section-title">DRIVE CONTROL</div>
      <div class="dpad-wrap">
        <div class="dpad">
          <div class="btn btn-empty"></div>
          <div class="btn" id="btn-w"
              onmousedown="press('w')" onmouseup="release()"
              ontouchstart="press('w')" ontouchend="release()">
            <i class="fas fa-chevron-up"></i>
          </div>
          <div class="btn btn-empty"></div>

          <div class="btn" id="btn-a"
              onmousedown="press('a')" onmouseup="release()"
              ontouchstart="press('a')" ontouchend="release()">
            <i class="fas fa-chevron-left"></i>
          </div>
          <div class="btn btn-stop" id="btn-s" onclick="press('s')">
            <i class="fas fa-stop"></i>
          </div>
          <div class="btn" id="btn-d"
              onmousedown="press('d')" onmouseup="release()"
              ontouchstart="press('d')" ontouchend="release()">
            <i class="fas fa-chevron-right"></i>
          </div>

          <div class="btn btn-empty"></div>
```

```
            <div class="btn" id="btn-x"
                onmousedown="press('x')" onmouseup="release()"
                ontouchstart="press('x')" ontouchend="release()">
               <i class="fas fa-chevron-down"></i>
            </div>
            <div class="btn btn-empty"></div>
         </div>
      </div>
   </div>

   <!-- Manual Telemetry -->
   <div class="card">
      <div class="section-title">TELEMETRY</div>
      <div class="telemetry">
         <div class="tele-item">
            <div class="tele-label">STATUS</div>
            <div class="tele-val" id="tele-status">IDLE</div>
         </div>
         <div class="tele-item">
            <div class="tele-label">DIRECTION</div>
            <div class="tele-val" id="tele-dir">—</div>
         </div>
         <div class="tele-item">
            <div class="tele-label">LEFT MOTOR</div>
            <div class="tele-val" id="tele-lm">OFF</div>
         </div>
         <div class="tele-item">
            <div class="tele-label">RIGHT MOTOR</div>
            <div class="tele-val" id="tele-rm">OFF</div>
         </div>
         <div class="tele-item">
            <div class="tele-label">SPEED</div>
            <div class="tele-val" id="tele-speed">20%</div>
         </div>
         <div class="tele-item">
            <div class="tele-label">PING</div>
            <div class="tele-val" id="tele-ping">—</div>
         </div>
      </div>
   </div>

</div><!-- /panel-manual -->

<!-- ═══ AUTO PANEL ═══ -->
<div class="panel-auto">
```

```
      <!-- Speed (auto still allows speed adjustment) -->
      <div class="card">
        <div class="speed-header">
          <div class="section-title" style="margin:0">AUTO SPEED</div>
          <div class="speed-val"><span id="auto-speed-
display">20</span>%</div>
        </div>
        <input type="range" min="10" max="100" value="20" id="auto-speed-
slider">
      </div>

      <!-- Auto Telemetry -->
      <div class="card">
        <div class="section-title">AUTO TELEMETRY</div>
        <div class="telemetry">
          <div class="tele-item">
            <div class="tele-label">DETECTOR</div>
            <div class="tele-val grn" id="auto-det-mode">SEARCH</div>
          </div>
          <div class="tele-item">
            <div class="tele-label">ZONE</div>
            <div class="tele-val grn" id="auto-zone">—</div>
          </div>
          <div class="tele-item">
            <div class="tele-label">ACTION</div>
            <div class="tele-val grn" id="auto-action">STOPPED</div>
          </div>
          <div class="tele-item">
            <div class="tele-label">CAMERA FPS</div>
            <div class="tele-val grn" id="auto-fps">—</div>
          </div>
          <div class="tele-item">
            <div class="tele-label">DETECT MS</div>
            <div class="tele-val grn" id="auto-det-ms">—</div>
          </div>
          <div class="tele-item">
            <div class="tele-label">PING</div>
            <div class="tele-val grn" id="auto-ping">—</div>
          </div>
        </div>
      </div>

    </div><!-- /panel-auto -->

</main>
```

```
<div class="statusbar">
  <span>PIBOT v3.0</span>
  <span id="status-text">WAITING...</span>
  <span id="clock">--:--:--</span>
</div>

<script>
  // ── Native WebSocket — no socket.io overhead ──────────────────
  const _wsURL = 'ws://' + window.location.hostname + ':5001';
  let ws, wsReady = false;
  let currentMode = 'manual';
  let currentDir  = null;
  let _pingStart  = 0;

  function wsConnect() {
    ws = new WebSocket(_wsURL);

    ws.onopen = () => {
      wsReady = true;
      document.getElementById('conn-dot').classList.add('connected');
      document.getElementById('conn-label').textContent = 'ONLINE';
      setStatus('CONNECTED');
    };

    ws.onclose = () => {
      wsReady = false;
      document.getElementById('conn-dot').classList.remove('connected');
      document.getElementById('conn-label').textContent = 'OFFLINE';
      setStatus('RECONNECTING...');
      setTimeout(wsConnect, 2000);   // auto-reconnect
    };

    ws.onerror = () => {
      setStatus('WS ERROR');
    };

    ws.onmessage = (e) => {
      const msg = e.data;

      if (msg === 'PONG') {
        const ms = Math.round(performance.now() - _pingStart);
        document.getElementById('tele-ping').textContent = ms + ' ms';
        document.getElementById('auto-ping').textContent = ms + ' ms';
        return;
      }
```

```
        if (msg.startsWith('HELLO:')) {
            // Parse initial state: HELLO:mode=manual:net=True
            const parts = msg.split(':');
            parts.forEach(p => {
                if (p.startsWith('mode=')) setModeUI(p.split('=')[1]);
            });
            return;
        }

        if (msg.startsWith('MODE:')) {
            setModeUI(msg.split(':')[1]);
            return;
        }

        if (msg.startsWith('AUTO:')) {
            // AUTO:zone:det_mode:fps:detect_ms
            const [, zone, det_mode, fps, det_ms] = msg.split(':');
            document.getElementById('auto-det-mode').textContent =
det_mode.toUpperCase();
            document.getElementById('auto-zone').textContent = zone;
            const actions = {LEFT:'TURN LEFT', MIDDLE:'FORWARD',
RIGHT:'TURN RIGHT', NONE:'STOPPED'};
            document.getElementById('auto-action').textContent = actions[zone] ||
'STOPPED';
            document.getElementById('auto-fps').textContent = fps + ' fps';
            document.getElementById('auto-det-ms').textContent = det_ms + ' ms';

            ['zone-left','zone-mid','zone-right'].forEach(id =>
                document.getElementById(id).className = 'zone-cell');
            if (zone==='LEFT') document.getElementById('zone-
left').classList.add('active-left');
            if (zone==='MIDDLE') document.getElementById('zone-
mid').classList.add('active-mid');
            if (zone==='RIGHT') document.getElementById('zone-
right').classList.add('active-right');
        }
    };
}

function wsSend(msg) {
    if (ws && wsReady) ws.send(msg);
}

wsConnect();
```

```
  // ── Ping every 2 s ──────────────────────────────────────────────
  setInterval(() => {
    if (!wsReady) return;
    _pingStart = performance.now();
    wsSend('PING');
  }, 2000);

  // ── Heartbeat: re-send held direction every 3 s ─────────────────
  setInterval(() => {
    if (currentDir && currentDir !== 's' && currentMode === 'manual') {
      wsSend('DRIVE:' + currentDir);
    }
  }, 3000);

  // ── Mode switching ──────────────────────────────────────────────
  function setModeUI(mode) {
    currentMode = mode;
    document.body.className = 'mode-' + mode;
    document.getElementById('btn-mode-manual').className =
      'mode-btn' + (mode === 'manual' ? ' active-manual' : '');
    document.getElementById('btn-mode-auto').className =
      'mode-btn' + (mode === 'auto' ? ' active-auto' : '');
    document.getElementById('zone-bar').style.display =
      mode === 'auto' ? 'grid' : 'none';
    if (mode === 'manual') { clearActive(); currentDir = null; }
    setStatus(mode === 'auto' ? 'AUTO MODE — TRACKING' : 'MANUAL
MODE');
  }

  function setMode(mode) {
    setModeUI(mode);
    wsSend('MODE:' + mode);
  }

  // ── Speed sliders ───────────────────────────────────────────────
  document.getElementById('speed-slider').addEventListener('input', function () {
    const val = this.value;
    document.getElementById('speed-display').textContent = val;
    document.getElementById('tele-speed').textContent    = val + '%';
```

```
    wsSend('SPEED:' + val);
  });
  document.getElementById('auto-speed-slider').addEventListener('input', function
() {
    const val = this.value;
    document.getElementById('auto-speed-display').textContent = val;
    wsSend('SPEED:' + val);
  });

  // ── Manual drive
──────────────────────────────────────────────────────────────────────────
  const dirNames = {w:'FORWARD',x:'REVERSE',a:'LEFT',d:'RIGHT',s:'STOP'};
  const allBtns  = ['btn-w','btn-a','btn-d','btn-x','btn-s'];

  function clearActive() {
    allBtns.forEach(id =>
document.getElementById(id)?.classList.remove('active'));
  }

  function press(dir) {
    if (currentMode !== 'manual') return;
    if (dir === currentDir) return;
    currentDir = dir;
    clearActive();
    document.getElementById('btn-' + dir)?.classList.add('active');
    wsSend('DRIVE:' + dir);
    updateManualTelemetry(dir);
  }
  function release() {
    if (currentMode !== 'manual') return;
    currentDir = null;
    clearActive();
    wsSend('DRIVE:s');
    updateManualTelemetry('s');
  }

  function updateManualTelemetry(dir) {
    document.getElementById('tele-dir').textContent = dir==='s' ? '—' :
dirNames[dir];
    document.getElementById('tele-status').textContent = dir==='s' ? 'IDLE' :
'MOVING';
    document.getElementById('tele-lm').textContent = (['w','a','x'].includes(dir)) ?
'ON' : 'OFF';
    document.getElementById('tele-rm').textContent = (['w','d','x'].includes(dir)) ?
'ON' : 'OFF';
```

```
  }

  // ── Keyboard ──────────────────────────────────────────────────────

  const validKeys = new Set(['w','a','s','d','x']);
  document.addEventListener('keydown', e => {
    const k = e.key.toLowerCase();
    if (validKeys.has(k) && !e.repeat) press(k);
  });
  document.addEventListener('keyup', e => {
    const k = e.key.toLowerCase();
    if (k !== 's' && validKeys.has(k)) release();
  });

  // ── Helpers ───────────────────────────────────────────────────────

  function setStatus(msg) {
    document.getElementById('status-text').textContent = msg;
  }

  setInterval(() => {
    document.getElementById('clock').textContent =
      new Date().toLocaleTimeString('en-GB');
  }, 1000);
</script>

</body>
</html>
```