
# FOSSEE Semester Long Internship Report

On

## Design, Iterative Prototyping and Deployment of an Autonomous Guided Vehicle for Industrial Material Handling

**Submitted by Shubham Dombale**
*4th Year, Department of Computer Engineering*
*Vidyalankar Institute of Technology, Mumbai*

**Under the Guidance of**
**Prof. Jayendran Venkateswaran**
*Department of IEOR*
*Indian Institute of Technology Bombay*



15 May 2026

# Acknowledgments

I would like to express my sincere gratitude to everyone who supported and guided me throughout the course of my internship and the successful completion of this project. Their encouragement, mentorship, and constructive feedback were instrumental in shaping my learning experience and strengthening my technical understanding.

I am deeply thankful to the **FOSSEE (Free/Libre and Open Source Software for Education)** team for providing me with this valuable opportunity to work on industrial engineering research projects. The internship offered a highly enriching environment that allowed me to apply theoretical knowledge to practical engineering challenges.

I would like to express my sincere appreciation to **Prof. Jayendran Venkateswaran**, **Department of IEOR**, **Indian Institute of Technology Bombay**, for his continuous guidance, mentorship, and technical insights throughout the internship. His direction helped me approach problems with structured engineering thinking and clarity.

I am also grateful to the **M.Tech mentors** who worked closely with me during the internship. Their technical discussions, suggestions, and feedback significantly contributed to the successful completion of the project.

I would further like to thank **Mr.Sumanto Kar** for his support and coordination during the internship period. His guidance and encouragement played an important role in ensuring smooth progress throughout the internship.

I gratefully acknowledge the support of the **National Mission on Education through Information and Communication Technology (ICT)**, **Ministry of Education (MoE)**, **Government of India**, for facilitating this project and enabling such valuable learning opportunities.

Finally, I extend my sincere thanks to my institution, **Vidyalankar Institute of Technology**, and its faculty members for their continued encouragement and support.

# Contents

# Chapter 1

# Introduction

## 1.1 National Mission on Education through ICT

The National Mission on Education through ICT (NMEICT) is a centrally sponsored scheme under the Department of Higher Education, Ministry of Education (MoE), Government of India. It was established to leverage the transformative potential of Information and Communication Technology (ICT) to enhance teaching and learning processes across Higher Education Institutions. The mission operates on an anytime-anywhere learning paradigm and aligns strictly with the three cardinal principles of the National Education Policy: access, equity, and quality.

To achieve these goals, the mission focuses on two major components: providing affordable access devices and connectivity to all learners and institutions, and generating high-quality e-content free of cost. NMEICT aims to seamlessly bridge the digital divide, ensuring that students and teachers in both urban and rural areas are empowered to participate in the modern knowledge economy. Its key focus areas include the development of e-learning pedagogies, the establishment of virtual laboratories, and the facilitation of online testing and certification through accessible platforms like EduSAT and Direct-to-Home (DTH) networks. Furthermore, it heavily emphasizes training teachers to integrate ICT-based methodologies into their traditional teaching frameworks. For more details, the official portal is accessible at www.nmeict.ac.in.

### 1.1.1 ICT Initiatives of the Ministry of Education

Under the NMEICT umbrella, the Ministry of Education has launched numerous specialized initiatives targeting students, researchers, and academic institutions. Rather than relying on physical infrastructure alone, these initiatives provide scalable digital resources:

- **SWAYAM & SWAYAMPRABHA:** SWAYAM allows students to earn academic credits via structured online courses, while institutions can host these courses and accept the credits. SWAYAMPRABHA complements this by broadcasting 24x7 educational TV programs for continuous learning.
- **National Digital Library (NDL) & e-PG Pathshala:** These platforms provide vast repositories of e-content across multiple disciplines. Students gain free access to textbooks and full-text e-resources, while institutions can list their e-content and form NDL Clubs.
- **Shodhganga & e-ShodhSindhu:** Aimed directly at the research community, Shodhganga provides access to Indian research theses, and e-ShodhSindhu provides institutional access to vital e-resources.
- **Hands-on Learning Platforms:** Initiatives like **e-Yantra** provide hands-on training in embedded systems, allowing institutions to create dedicated labs in collaboration with IIT Bombay. Similarly, **Virtual Labs** allow students to perform curriculum-based online experiments without needing physical lab equipment.
- **FOSSEE & Spoken Tutorial:** These initiatives promote self-learning of IT skills and encourage volunteering for open-source software development. They empower institutions to run modern technical labs without the financial burden of proprietary software licenses.

## 1.2 The FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project is a flagship initiative specifically designed to promote the use of open-source tools in academia and research. Sponsored by the NMEICT and the Ministry of Education, FOSSEE aims to reduce the dependency of Indian educational institutions on expensive, proprietary software.

### 1.2.1 Projects and Core Activities

FOSSEE supports the widespread adoption of various open-source alternatives to enhance the quality of technical education. Its core activities are divided into several strategic areas:

- **Textbook Companion Project:** FOSSEE volunteers port solved examples from standard engineering and science textbooks into open-source software environments, creating a massive repository of accessible reference material.
- **Lab Migration:** The project actively facilitates and guides institutions in migrating their existing proprietary software labs to Free/Libre and Open Source Software (FLOSS) alternatives.
- **Niche Software Promotion:** FOSSEE undertakes specialized campaigns to promote highly specific, niche software tools that are critical for advanced engineering research but are traditionally locked behind expensive licenses.
- **Community Building:** FOSSEE hosts user forums and organizes regular workshops and conferences to train students and faculty, fostering a collaborative, self-sustaining ecosystem of open-source developers and users.

### 1.2.2 Fellowships and Internships

To drive continuous engagement, FOSSEE offers various competitive internship and fellowship opportunities, including the Winter Internship, Summer Fellowship, and Semester-Long Internships.

These programs are uniquely accessible; students from any degree background and academic stage are eligible to apply. The selection process is strictly merit-based, relying on the completion of rigorous screening tasks. These tasks typically involve programming, scientific computing, or data collection exercises designed to directly benefit the broader FLOSS community. Candidates are expected to complete these tasks within a tight one-week timeframe, testing both their technical proficiency and their ability to deliver under pressure. Further information regarding these opportunities is maintained on the official FOSSEE website.

# Chapter 2

# Automated Guided Vehicles: Design, Development and Deployment

## 2.1 Introduction to Automated Guided Vehicles

The advent of **Industry 4.0** has necessitated a radical shift in how manufacturing floors, warehouses, and distribution hubs handle the movement of raw material, work-in-progress assemblies, and finished goods. Sitting at the centre of this shift is the **Automated Guided Vehicle (AGV)** – a self-propelled, computer-controlled load carrier that moves along the facility floor without an onboard human driver. Unlike a conventional forklift or hand-truck, an AGV is a closed-loop mechatronic system: it perceives its position relative to a defined path, computes a corrective steering response, and actuates its own drivetrain, all without continuous human supervision.

The commercial appeal of the AGV is straightforward. By automating repetitive point-to-point transport tasks, an AGV fleet reduces the direct labour cost associated with material movement, removes a major source of workplace collisions and repetitive-strain injury, and – perhaps most importantly for flexible manufacturing – allows the physical layout of a factory to be reorganised without ripping out fixed conveyor infrastructure. This report documents the design, iterative failure analysis, and eventual maturation of an AGV platform developed as part of a FOSSEE-sponsored internship at IIT Bombay, culminating in a full-scale, payload-handling final prototype capable of autonomous pick-up and drop-off operations.

## 2.2 Historical Background and Evolution

The lineage of the modern AGV can be traced back to **1953**, when **Barrett Electronics Corporation**, under inventor **A.M. Barrett Jr.**, converted a conventional towing tractor into a vehicle that followed a current-carrying wire buried in a shallow trench cut into the factory floor. The wire radiated a weak magnetic field that a coil-based sensor on the tractor could track, allowing the vehicle to hold a fixed route without a driver. This single invention is widely credited with founding the entire AGV industry.

Through the 1970s, the arrival of solid-state electronics allowed AGVs to shrink dramatically in size and cost while becoming markedly more dependable. **Volvo** is frequently cited as a pioneer of this era for deploying AGVs as *mobile assembly platforms* at its Kalmar, Sweden plant, where individual car bodies rode on dedicated AGVs that moved between workstations, replacing the rigid, single-speed pace of the traditional moving assembly line with a more flexible, cellular manufacturing philosophy.

The 1980s and 1990s introduced laser-based guidance and embedded microprocessors capable of running real-time dispatching and collision-avoidance algorithms across an entire AGV fleet. Today, that eight-decade lineage has converged with **Artificial Intelligence (AI)** and **Simultaneous Localization and Mapping (SLAM)**, giving rise to AGVs (frequently re-branded as Autonomous Mobile Robots, or AMRs) that require no fixed infrastructure at all and can re-route themselves dynamically around obstacles and even other robots.

## 2.3 Core Components of an AGV System

Regardless of scale, every functioning AGV is an integration of five interdependent subsystems:

- **Chassis and Mechanical Framework:** The structural skeleton of the vehicle. It must resist bending and torsional loads introduced by an off-centre payload while keeping the vehicle's centre of gravity low enough to avoid tipping during a turn.
- **Propulsion and Actuation System:** The motors, gearboxes and drive wheels that convert electrical energy into controlled linear and rotational motion. The system must supply enough starting torque to overcome static friction under full payload while still allowing fine velocity modulation for smooth cornering.

- **Navigation and Sensory Array:** The perception layer of the vehicle – typically a combination of a primary path-tracking sensor (magnetic tape reader, IR array, camera, or LiDAR) and a secondary safety layer (ultrasonic or infrared proximity sensors, bumper switches, emergency stops).
- **Energy Storage and Power Management:** The onboard battery bank – lead-acid, Li-ion, or LiPo – along with the voltage regulation circuitry required to safely power both the high-current drivetrain and the low-current logic electronics from a common source.
- **Onboard Supervisory Controller:** The computational core – a microcontroller or industrial PC – that continuously ingests sensor data, executes the control law (commonly a PID loop), and issues commands to the motor drivers.

## 2.4 Classification of AGVs

AGVs are rarely built to a single template; their form factor is dictated almost entirely by the payload and process they are meant to serve.

### 2.4.1 Tow Vehicles (Tuggers)

Among the earliest AGV form factors, tuggers pull a train of unpowered trailers rather than carrying the load directly. This makes them highly efficient for moving large volumes of raw material in a single trip from a warehouse to a production line.

### 2.4.2 Unit Load Carriers

These vehicles carry a single discrete load directly on an onboard deck, sometimes equipped with a small conveyor or lift table to automate the loading and unloading process at each stop. The final prototype developed in this project belongs to this category.

### 2.4.3 Automated Forklifts

Designed to fully displace a human-driven forklift, these AGVs feature automated masts and forks capable of placing and retrieving pallets from multi-level warehouse racking.

### 2.4.4 Custom Assembly AGVs

Common in the automotive and aerospace industries, these purpose-built platforms carry a product through successive stages of assembly, effectively functioning as a mobile, decentralised assembly line.

## 2.5 Navigation and Pathfinding Technologies

The navigation method an AGV relies on defines both its installation cost and its flexibility once deployed:

- **Magnetic Tape and Wire Guidance:** The oldest and most mechanically robust method. A magnetic sensor bank follows a strip of tape or a buried wire; the trade-off is that changing the route means physically altering the floor.
- **Laser Target Navigation:** Reflective targets mounted around the facility are triangulated by a rotating on-board laser scanner to compute an absolute vehicle position.
- **Vision-Based Navigation:** Stereo cameras and computer-vision pipelines (e.g. OpenCV) allow the vehicle to follow painted lines or recognise environmental landmarks.
- **Natural Feature SLAM:** The current state of the art. 2D or 3D LiDAR is used to build a live map of the environment while simultaneously planning a path through it, removing the need for any painted line, tape, or wire.

## 2.6 Project Development Methodology

This project deliberately followed a three-phase, iterative hardware methodology rather than attempting a single, fully-specified final design from the outset. Each phase was treated as a disposable proof-of-concept whose sole purpose was to validate (or invalidate) one specific engineering hypothesis – sensor architecture in Phase 1, mechanical load-bearing capacity in Phase 2, and full industrial functionality (structural rigidity, high-current propulsion, and an automated pick-and-place payload interface) in Phase 3. This mirrors accepted rapid-prototyping practice in mechatronic product development, where the cost of discovering a design flaw rises by an order of magnitude with every stage the flaw survives undetected. A secondary, and

equally important, objective that shaped every phase of this project was **educational accessibility**: every component selected across all three prototypes is an inexpensive, widely available "hobbyist-grade" part, so that the resulting platform can be reproduced by an engineering student, or replicated as a teaching tool in an under-resourced laboratory, without requiring an industrial parts budget.

## 2.7 Phase 1: Breadboard Proof-of-Concept

Before any heavy fabrication was attempted, a lightweight, breadboard-based prototype was built to answer three narrow engineering questions: could an 8-channel analog IR array give sufficiently fine-grained positional resolution to drive a smooth PID loop; could a single 8-bit microcontroller run that PID loop *and* service a secondary manual-override input without missing cycles; and what fault-recovery behaviour was needed when the sensor array lost the line entirely.

### 2.7.1 Hardware Architecture and Component Integration

The Phase 1 chassis was a lightweight, laser-cut planar rapid-prototyping platform. Every component on it was chosen to minimise wiring complexity and preserve the largest possible number of usable I/O pins:

- **Microcontroller Processing Unit (Arduino Nano):** Selected over the more common Arduino Uno for two concrete reasons. Its DIP-style pin header allows it to sit directly on a mini breadboard, and its ATmega328P exposes a full 8-channel ADC (A0–A7) rather than the Uno's 6, which is a hard requirement for reading all eight channels of the analog IR array without external multiplexing.

- **Propulsion:** Two N20 micro metal-gear motors paired with 5 cm high-traction rubber tyres. The N20 was chosen for its favourable torque-to-size ratio and predictable low-RPM behaviour, avoiding the large stall currents that could brown-out a breadboard-powered Nano.

- **Motor Driver (L298N H-Bridge):** A standard dual H-bridge module that lets the Nano set the direction of each motor through logic pins (IN1–IN4) and modulate speed through PWM on the enable pins (ENA/ENB).

- **Navigation Sensor Array:** An 8-channel analog IR reflectance array mounted at the leading edge of the chassis. Because each channel outputs a continuous voltage rather than a binary HIGH/LOW, the array can resolve exactly *where* between two adjacent sensors the tracking line sits, rather than only which single sensor currently sees it.

- **Power Distribution:** Three 18650 Li-ion cells in series (nominal 11.1 V, peak 12.6 V) fed the L298N directly for the drivetrain, while

the Nano's onboard regulator stepped the same rail down to a stable 5 V logic supply through its VIN pin.

### 2.7.2 Dual-Mode Operational Logic

To emulate the way a real factory-floor AGV must occasionally be handed back to a human operator – for an emergency stop or an unplanned docking manoeuvre – Phase 1's firmware supported two mutually exclusive operating modes.

#### 1. Teleoperation (Manual Override) Mode

A VS1838B infrared receiver, decoding the standard NEC protocol from a handheld remote, allowed an operator to directly command the drive-train. Received hex codes were mapped onto three kinematic behaviours: synchronous forward/reverse motion on both motors, differential turning by halting one motor while driving the other, and zero-radius pivot turning by spinning the two motors in opposite directions.


Figure 2.1: The handheld IR remote used for Phase 1 manual teleoperation, alongside the breadboard-mounted Arduino Nano and L298N motor driver.

#### 2. Autonomous Line-Following Mode

When switched into autonomous mode, the firmware abandoned the IR-remote polling loop entirely and dedicated every CPU cycle to navigation. All eight ADC channels (A0–A7) were sampled at a high refresh rate, and a weighted average of the eight readings was used to compute a single continuous positional-error value describing exactly how far the tracking tape had drifted from the chassis centreline.

Steering directly off this raw error value produces a well-known failure mode called *hunting*, in which the vehicle zig-zags violently across the tape

instead of tracking it smoothly. To eliminate hunting, a **Proportional-Integral-Derivative (PID)** controller was layered on top of the raw positional error:

- **Proportional (P):** An immediate steering response scaled directly by the current positional error.
- **Integral (I):** An accumulator of past error, used to correct any persistent mechanical bias (e.g. one gearbox running slightly slower than the other).
- **Derivative (D):** A predictive term that dampens the steering response as the error approaches zero, preventing the vehicle from overshooting the centreline on sharp curves.

The full mathematical derivation of the PID law, its discrete-time implementation, and the empirical process used to tune $K_p$, $K_i$, and $K_d$ are deferred to Section 2.9.10, where the same control law re-appears – considerably re-tuned – inside the firmware of the final, full-scale prototype.

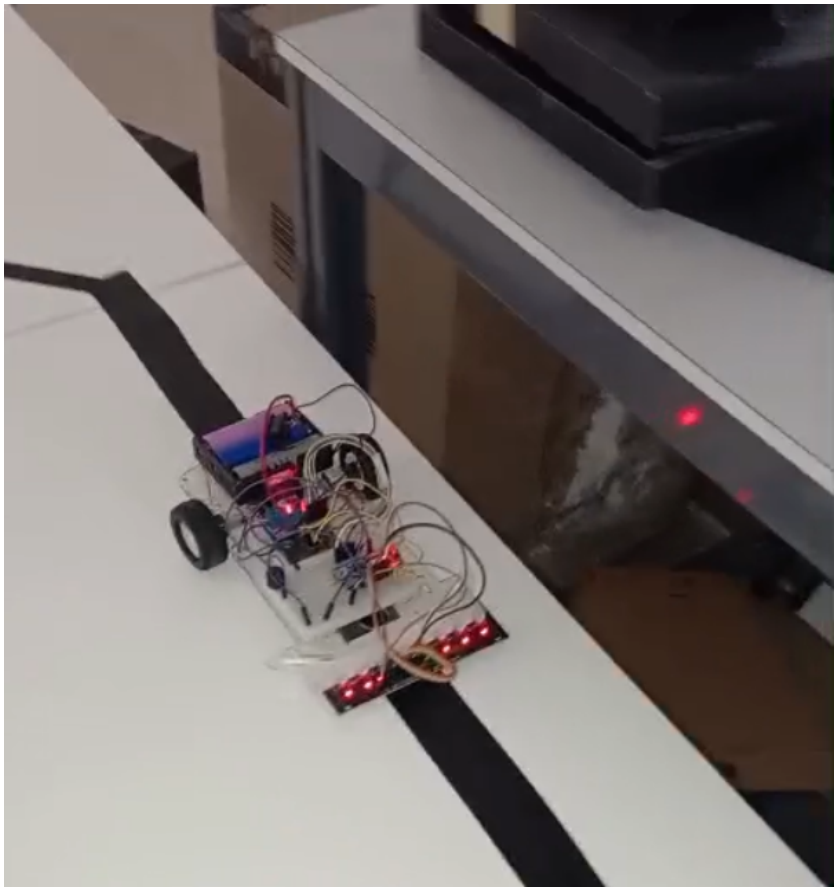

Figure 2.2: The Phase 1 prototype operating in autonomous line-following mode, tracking a dark path with its front-mounted 8-channel analog IR array.

### 2.7.3 Fault-Tolerance: Autonomous Line Recovery Algorithm

An unrecoverable derailment is unacceptable on a real factory floor, so Phase 1 introduced a dedicated recovery behaviour for the moment the sensor array loses the tracking line entirely (all eight channels simultaneously reading the high-reflectance white floor). On detecting this condition, the firmware executed a four-step **Line Recovery Protocol**: an immediate halt of forward motion; a slow, zero-radius pivot sweep in the direction the line was last seen; continuous re-scanning during the sweep for the sharp reflectance drop that indicates the black line has been re-acquired; and finally, a clean resumption of normal PID-controlled forward tracking.

### 2.7.4 Acoustic Diagnostic Feedback System

Because a factory floor is typically too loud for a status LED to be a useful diagnostic channel, a passive buzzer was added to the Nano's digital outputs and driven with software-generated PWM tones for two distinct purposes: a loud, sustained alarm tone if the 360-degree recovery sweep timed out without reacquiring the line, and a short melodic chime on boot-up, used purely to confirm that the buzzer circuit and the microcontroller's internal timers were both functioning correctly before a run began.

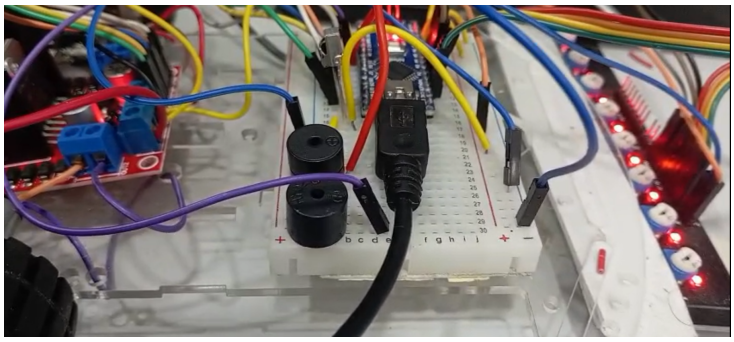

Figure 2.3: The dual passive buzzers on the Phase 1 breadboard, used for PWM-based auditory state feedback and fault alarms.

**Conclusion of Phase 1:** The breadboard prototype successfully answered all three questions it was built to test – the 8-channel analog array gave more than enough resolution for smooth PID tracking, the Nano handled both operating modes without any observable timing conflict, and the recovery sweep reliably reacquired the line after a deliberate derailment. What it categorically could *not* do was carry anything. The flat acrylic chassis had no payload containment and no structural rigidity to speak of, meaning it validated the sensing and control electronics but said nothing about whether the same architecture could survive being scaled up into a genuine unit-load carrier. That question became the focus of Phase 2.

## 2.8 Phase 2: Bin-Chassis Unit-Load Prototype

The goal of Phase 2 was narrow and specific: keep the Phase 1 electronics architecture completely unchanged, and re-house it inside a chassis that could actually contain and transport a physical payload.

### 2.8.1 Material Selection: The Bin-Chassis Concept

Rather than fabricating a bespoke sheet-metal enclosure – a route that would have added weight, cost, and manufacturing lead time – an off-the-shelf, ribbed industrial storage bin was repurposed as the vehicle's monocoque chassis. This delivered several mechanical benefits essentially for free: the bin's High-Density Polyethylene (HDPE) construction gave an excellent strength-to-weight ratio well within the pulling capacity of the N20 motors; its moulded-in vertical ribbing acted as a stiffener against torsional flex under an unevenly distributed payload; and its deep walls naturally contained the payload, preventing it from sliding out during sharp turns or sudden stops.

### 2.8.2 Chassis Modification and Hardware Mounting

Converting the passive bin into an active chassis required precise subtractive manufacturing. Rectangular cut-outs were milled into the lower side walls so the N20 motors could be flush-mounted with their wheel axles aligned to the bin's centre of gravity – a mid-drive layout that maximises traction by ensuring payload weight presses straight down onto the drive wheels. A second aperture was cut into the forward floor panel to give the IR array an unobstructed, millimetre-tolerance line of sight to the floor while still protecting its PCB from debris impacts.

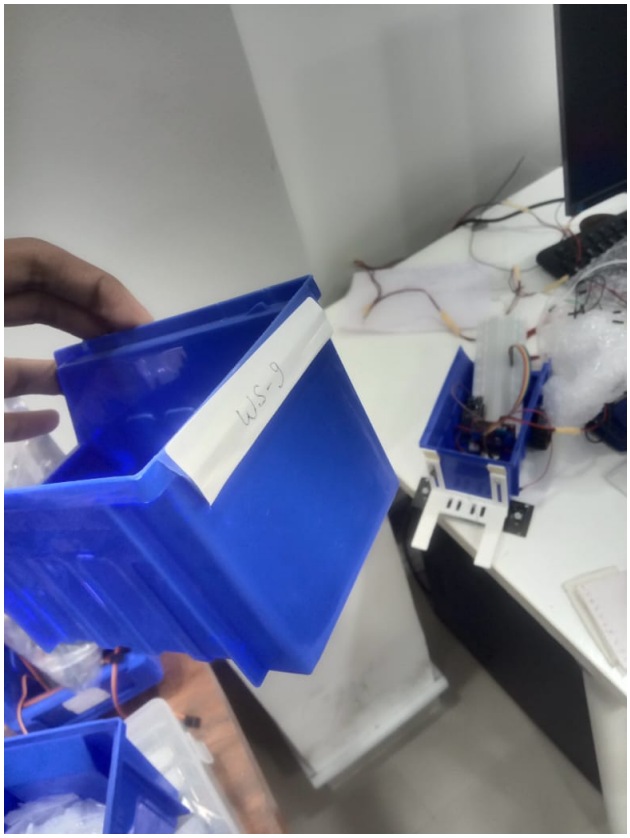


Figure 2.4: The Phase 2 storage bin during conversion into a monocoque chassis, showing the rigid ribbed sidewalls that would house the electronics and carry the payload.

### 2.8.3 Internal Decking and Payload Isolation Layer

Simply dropping a payload into the bin risked crushing the Nano or short-circuiting the battery terminals, so a dual-layer internal architecture was introduced. The lower volume of the bin – the *avionics bay* – housed the battery pack, the L298N driver, the Nano, and all wiring. A custom-fabricated false floor, mounted horizontally above the highest electronic component, created a second, upper volume – the *payload deck* – onto which blocks, tools, or parts bins could be loaded without any risk to the circuitry beneath. As a secondary benefit, this deck also spread the payload's mechanical stress evenly across the bin's reinforced sidewalls rather than concentrating it on the thin plastic floor.

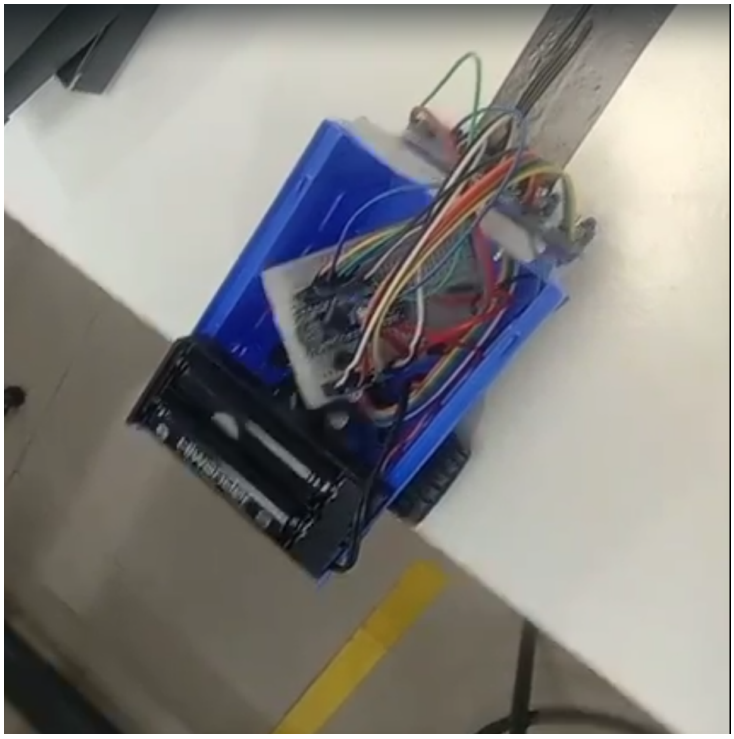

Figure 2.5: Underside view of the Phase 2 chassis, showing the externally mounted N20 gear motors, the front passive caster wheel, and the floor-level IR sensor array.

**Conclusion of Phase 2:** The bin-chassis approach worked – the vehicle could now carry a genuine payload while running the exact same PID and recovery firmware validated in Phase 1. But two limitations became obvious as soon as heavier and less evenly distributed loads were tested. First, HDPE is not a dimensionally stiff material under sustained load; the bin walls exhibited a small but measurable amount of flex that, at higher speeds, translated into inconsistent IR sensor stand-off height. Second, and more importantly, Phase 2 was still a pure *transport* vehicle – an operator (or a second machine) still had to physically place the payload onto the deck and remove it at the destination. Neither prototype attempted to automate the loading and unloading step itself. Both of these gaps – structural rigidity under load, and a genuinely automated payload interface – became the explicit design targets for the third and final prototype.

## 2.9 Phase 3: Final Prototype – The 500×500 mm Pick-and-Place Industrial AGV

### 2.9.1 Design Motivation and Educational Objectives

The final prototype was conceived to close both gaps left open by Phase 2, while holding firm to the project's underlying educational mandate. Structurally, the chassis needed to be rebuilt in a material stiff enough to guarantee that the IR array's stand-off height above the floor – and therefore its calibration – would not drift under load or vibration. Functionally, the vehicle needed a genuinely automated payload interface: rather than depending on a human to load and unload each trip, the AGV itself needed to detect a pick-up point, physically lift the payload onto its own deck, transport it, and lower it again at a drop-off point, entirely without manual intervention. At the same time, every component chosen for this build remains inexpensive, hobbyist-accessible hardware – a NEMA 17 stepper, an Arduino Nano, off-the-shelf H-bridge modules, ultrasonic range sensors, and 3D-printed brackets – so that the completed vehicle continues to serve its original purpose as a low-cost, reproducible **educational AGV platform**, giving a student hands-on exposure to differential-drive kinematics, PID control, stepper-driven linear actuation, and multi-sensor fusion, all on a single accessible build.

The result is a substantially larger vehicle than either of its predecessors: a 500 mm × 500 mm footprint built from structural aluminium extrusion, capable of autonomously locating a pick-up station purely from the line-tracking tape pattern, lifting a payload of up to **1 kg** onto its deck using an onboard lead-screw actuator, transporting it along the line to a drop-off station, and lowering it automatically – all while continuously watching for obstacles in its path.

### 2.9.2 Mechanical Architecture: 2020 Aluminium Extrusion Chassis

The structural frame of the final prototype was built entirely from **2020-series aluminium T-slot extrusion** (20 mm × 20 mm cross-section), the same profile family used in the Phase 3 conveyor prototypes documented elsewhere in this internship and in desktop 3D-printer gantries. Compared to the slotted-angle-iron and HDPE-bin chassis of the earlier prototypes, T-slot aluminium offers three decisive advantages at this larger scale:

- **Guaranteed Orthogonality:** Precision-cast 90-degree corner brack-

ets lock every rail junction into a true right angle, which is essential at a 500 mm span – any accumulated skew across a frame this size would be enough to misalign the drive-wheel axle relative to the IR array and destabilise the PID loop.

- **High Torsional Rigidity:** The internal ribbing of the extrusion strongly resists twisting under an off-centre payload, which keeps the sensor deck level and keeps the IR array's calibrated stand-off height constant regardless of how the 1 kg payload is distributed on the deck.
- **Infinite Linear Adjustability:** The continuous T-slot channel lets every bracket – motor mounts, caster mount, lift-column mounts, sensor mounts – be positioned and re-positioned along the rail before being locked down with M4 T-nuts, which proved invaluable during iterative tuning of the sensor stand-off height and the wheel track width.

The 500 mm × 500 mm footprint was chosen deliberately to be large enough to physically house the lead-screw lift column, the dual 12 V battery packs, and the electronics bay side-by-side without cramming them into the same vertical layer – directly addressing the internal packaging compromises the Phase 2 bin chassis was forced into.

### 2.9.3 Propulsion System: Dual Johnson Geared Motors and Caster Wheel

Where Phase 1 and Phase 2 relied on lightweight N20 micro-motors adequate only for their own tare weight, the final prototype's substantially larger frame, dual battery packs, and lead-screw lift column pushed the overall vehicle mass into a different regime entirely. The drivetrain was accordingly upgraded to **two 12 V Johnson-type DC geared motors**, each rated to comfortably move loads on the order of **10 kg**, giving the vehicle a very wide starting-torque margin over its actual laden weight. The two drive motors are mounted on a common axis roughly beneath the vehicle's centre of gravity, while a single large, freely castering wheel provides the third ground-contact point, completing a differential-drive tricycle layout identical in principle – if considerably larger in scale – to the layout already validated in Phases 1 and 2.

The differential-drive kinematics themselves are unchanged from the earlier prototypes: the two drive wheels are commanded independently, so that driving them at equal speed produces straight-line motion, driving them at unequal speeds produces a sweeping turn, and driving them at equal-and-opposite speed produces a zero-radius pivot on the spot. What *does* change

at this larger scale is the current draw required to actuate that kinematic model, which motivated the next major hardware upgrade.

### 2.9.4 Motor Driver: BTS7960 High-Current H-Bridge Modules

The L298N driver used in Phases 1 and 2 is a linear (not switching) H-bridge, and its internal transistors drop a non-trivial amount of voltage across themselves – acceptable for the sub-1 A currents drawn by the N20 motors, but a serious source of wasted power and heat when driving a 10 kg-rated Johnson motor under load. The final prototype therefore replaces the L298N with **two BTS7960 high-current H-bridge modules**, one per drive motor. Each BTS7960 module uses a pair of automotive-grade MOSFETs per leg rather than bipolar transistors, giving it a continuous current rating an order of magnitude beyond the L298N and a far lower on-resistance, which keeps the driver running cool even under sustained full-torque operation.

Electrically, each BTS7960 module exposes two independent PWM inputs – `RPWM` and `LPWM` – rather than the L298N's separate direction and enable pins. Applying a PWM signal to `RPWM` (with `LPWM` held low) drives the motor in one direction at a speed proportional to the duty cycle; applying PWM to `LPWM` instead reverses the motor. This is precisely the convention followed by the `setMotorSpeeds()` routine dissected in Section 2.9.10, where a signed speed value is decomposed into the correct `RPWM`/`LPWM` pair based on its sign.

### 2.9.5 Power System: Dual 12 V LiPo Battery Architecture

The final prototype is powered by **two independent 12 V, 2000 mAh, 35C-rated Lithium-Polymer battery packs**. Splitting the electrical system across two separate packs, rather than drawing everything from one larger pack, was a deliberate noise-isolation decision: the BTS7960 modules switch tens of watts of motor current at a high PWM frequency, and coupling that switching noise directly onto the same rail feeding the Arduino Nano's 10-bit ADC would corrupt the delicate analog readings coming from the IR sensor array. In the final architecture, one 12 V pack is dedicated to the high-current drive train – the two BTS7960 modules and the Johnson motors – while the second pack powers the logic and actuation rail: the Arduino Nano, the NEMA 17 stepper driver, the eight-channel IR array, the three ultrasonic modules, and the buzzer.

The 35C discharge rating of each pack is a significant safety margin at this

scale: a 2000 mAh cell rated at 35C can theoretically sustain a continuous discharge of up to 70 A, far beyond anything the drivetrain will draw even under a full-torque stall, which keeps the pack's internal voltage sag – and therefore the risk of a brown-out on the logic rail – to a minimum during the high-current pickup and drop manoeuvres.

### 2.9.6 Vertical Pick-and-Place Mechanism: NEMA 17 Stepper and Lead Screw

The single largest functional addition in Phase 3 is the automated loading mechanism, which is what elevates this build from a pure *transport* vehicle (Phases 1 and 2) to a genuine *pick-and-place* AGV. The mechanism is built around a **NEMA 17 hybrid stepper motor** driving a **lead screw and matching lead nut** – the same linear-actuation technology used almost universally in the Z-axis of desktop 3D printers – coupled to a 3D-printed carriage that rides on a pair of smooth steel guide rods via linear ball bearings.

Figure 2.6: Close-up of the Phase 3 lift mechanism: the NEMA 17 stepper (bottom) drives the lead screw, which raises and lowers the 3D-printed carriage plate riding on twin smooth guide rods via linear bearings.

A stepper motor was chosen over a simple geared DC motor for this specific axis because a stepper's position is inherently known in open loop – every electrical pulse sent to the driver corresponds to a fixed, repeatable angular increment of the motor shaft, and therefore a fixed, repeatable linear increment of the lead nut along the screw. This gives the lift axis far more predictable, jitter-free positioning than an equivalent DC-motor-driven mechanism would achieve without an encoder, at a fraction of the cost and wiring complexity of adding one.

Mechanically, the lead screw converts the stepper's rotary motion into linear motion of the carriage plate. As the screw rotates, the lead nut – rigidly fixed to the 3D-printed carriage – is forced to translate along the screw's axis, while the twin guide rods and their linear bearings absorb any radial or torsional load and prevent the carriage from rotating along with the screw. This is the same working principle, and largely the same physical hardware, used to raise and lower the print head on a FDM 3D printer's Z-axis – reapplied here to raise and lower a physical payload deck instead.

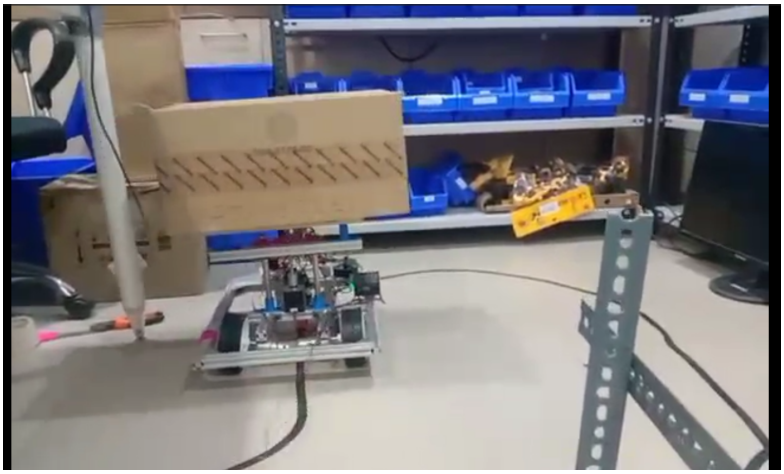

Figure 2.7: The Phase 3 prototype with the lift carriage raised, having lifted a boxed payload clear of the pick-up station during a live test run.

### 2.9.7 Homing and Limit-Switch Referencing

Because the stepper's position is only known *relative* to wherever it started – and not in any absolute sense – the firmware cannot simply assume the carriage begins each run at the bottom of its travel. To solve this, a mechanical **limit switch** is mounted at the lowest point of the carriage's travel. On every power-up, and again before every pickup and drop cycle, the firmware runs a dedicated **homing routine**: it drives the stepper continuously in the downward direction until the limit switch is physically triggered, at which point the stepper is immediately stopped and the carriage's current position is defined as the vehicle's zero-reference ("home") position. Every subsequent lift and lower motion is then executed relative to this freshly re-established home point, which prevents small positioning errors from accumulating silently across dozens of pickup/drop cycles over the course of an extended operating shift.

### 2.9.8 Line-Following Sensor Array and Automated Station Detection

The final prototype retains the same 8-channel analog IR reflectance array validated across both earlier prototypes, individually calibrated per channel to compensate for manufacturing variance in the LED/phototransistor pairs (a routine necessity in any array built from low-cost hobbyist IR sensors, since no two channels reach quite the same absolute reflectance value over black or white surfaces). What is entirely new in Phase 3 is the way the array's aggregate reading is used: rather than only computing a positional error for steering, the firmware also continuously evaluates the *sum* of the eight normalised sensor readings to recognise two special floor markings that were not present in either earlier prototype.

- **Pick-up Station – Solid Black Marker:** When all eight sensors simultaneously report a fully saturated black reading (i.e. the vehicle is sitting on top of a wide black tile or patch, rather than a narrow tracking strip), the firmware recognises this as a designated **pick-up point** and halts the drivetrain to begin the lift cycle described in Section 2.9.10.
- **Drop-off Station – Sustained White Marker:** Conversely, when all eight sensors report the high-reflectance white floor for a *sustained* number of consecutive control-loop iterations (rather than a brief, momentary gap), the firmware recognises this as the designated **drop-off point** and begins the lower cycle.

This is a deliberately elegant piece of systems design: it re-uses sensing hardware that was already present for line tracking, and re-purposes the exact same "line lost" detection logic that Phase 1 used purely for fault recovery, now giving it a second, productive meaning as an intentional station marker. The distinction between an accidental, momentary loss of the tracking tape (a small gap, a scuff, or a shadow) and a genuine, intentional drop-off station is made using a simple debounce counter – the white condition must persist for roughly fifty consecutive control-loop iterations before it is trusted as a real station, filtering out any brief, spurious reading.

### 2.9.9 Obstacle Avoidance Subsystem: Triple Ultrasonic Array and Buzzer

Neither earlier prototype had any awareness of what lay in its path beyond the tracking tape itself; both would happily drive straight into an obstacle

sitting on the line. The final prototype closes this gap with a forward-facing array of **three ultrasonic range sensors**, mounted left, centre, and right across the leading edge of the chassis, giving the vehicle a wide field of coverage across its own body width. Each sensor continuously measures the distance to the nearest object directly ahead of it; if any one of the three reports an object within a configured **15 cm** stop distance, the firmware immediately halts the drivetrain, sounds the buzzer as an audible warning to nearby personnel, and holds that stopped state for a brief, fixed interval before re-checking whether the path has cleared. This gives the AGV a basic but genuinely functional safety layer, appropriate to its role as an educational demonstrator operating in a shared lab space rather than a fully caged industrial cell.

### 2.9.10 Firmware Architecture and Supporting Libraries

The Phase 3 firmware, like its predecessors, is written in C++ on the Arduino framework, but its control-flow is considerably more elaborate: rather than a single flat loop with two branches (line-follow / recover), it must now arbitrate between four distinct behaviours – obstacle avoidance, line-following, pick-up-station handling, and drop-off-station handling – on every single pass of the main loop. Two external libraries were introduced specifically to support this expanded scope:

- **AccelStepper:** Used to drive the NEMA 17 lift stepper. Although the firmware in this build primarily uses AccelStepper's constant-speed `runSpeed()` mode rather than its full trapezoidal acceleration profile, the library still greatly simplifies generating clean STEP/DIR pulse trains without the main loop having to manage pulse timing by hand.
- **NewPing:** Used to drive all three ultrasonic sensors. NewPing handles the microsecond-level timing of the trigger pulse and echo measurement internally, and – as configured in this build – supports operating each sensor from a single shared I/O pin (used simultaneously as trigger and echo), which was an important pin-budget saving on a Nano that already has every other pin committed to the IR array, the drive motors, and the stepper.

#### Pin Mapping and Global Configuration

The firmware begins, exactly as in the earlier prototypes, by declaring the hardware interface as a set of named constants – a defensive habit that keeps

the physical wiring and the compiled logic in a single, unambiguous source of truth.

```
// ================ STEPPER PINS ================
#define STEP_PIN 13
#define DIR_PIN  12
#define LIMIT_SWITCH_PIN 3
#define BUZZER 2

AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
float stepperSpeed = 2000;

// ================ ULTRASONIC SENSORS ================
#define LEFT_PIN   4
#define MIDDLE_PIN 7
#define RIGHT_PIN  5
#define MAX_DISTANCE 50
#define STOP_DISTANCE 15

NewPing sonarLeft(LEFT_PIN, LEFT_PIN, MAX_DISTANCE);
NewPing sonarMiddle(MIDDLE_PIN, MIDDLE_PIN, MAX_DISTANCE);
NewPing sonarRight(RIGHT_PIN, RIGHT_PIN, MAX_DISTANCE);

// ================ MOTOR PINS ================
#define L_RPWM 9
#define L_LPWM 6
#define R_RPWM 11
#define R_LPWM 10

// ================ SENSORS ================
const int sensor_pins[8] = {A0,A1,A2,A3,A4,A5,A6,A7};
```

The `AccelStepper` object is constructed in `DRIVER` mode, meaning the library assumes an external stepper driver board (of the common A4988/DRV8825/TMC-family) is wired between the Nano and the NEMA 17, and that the Nano's only responsibility is to generate clean `STEP` and `DIR` logic signals – all current-regulation and microstepping is handled externally by the driver board itself, well outside the Nano's electrical capability. Note also that the three `NewPing` sensor objects are each instantiated with the *same* pin passed twice – once for trigger, once for echo – which is what allows all three ultrasonic modules to be wired using only a single Nano pin apiece.

### Calibration and PID Tuning Constants

The per-channel IR calibration array and the PID gains were both re-tuned from scratch for Phase 3, since neither the sensor stand-off height nor the vehicle's mass and inertia carried over unchanged from the earlier prototypes.

```
// ================ CALIBRATION ================
int white_value = 100;
int black_value[8] =
{
  500,500,550,550,
  600,700,700,620
};

// ================ PID ================
int base_speed = 90;
float kp = 10.0;
float ki = 0.0;
float kd = 35.0;

// ================ STATE ================
int weight[8] = {-7,-5,-3,-1,1,3,5,7};
float error=0,last_error=0,integral=0;
float last_side=0;

bool box_loaded=false;
int white_count=0;
```

Two changes from the Phase 1 tuning are worth calling out explicitly. First, the derivative gain $K_d = 35.0$ is more than double the Phase 1 value – a direct consequence of the final prototype's much greater physical mass and rotational inertia, which makes it slower to respond to a steering correction and therefore more prone to overshoot if the derivative term is not strengthened to compensate. Second, the base cruising speed was deliberately reduced to `base_speed = 90` (roughly a third of the earlier prototypes' cruising PWM), reflecting the simple physical reality that a heavier vehicle carrying a live payload needs a larger safety margin against skidding, tipping, or missing a station marker than a lightweight breadboard rig does. The integral gain remains at zero for the same anti-windup reasoning established in Phase 1 – accumulating a persistent steady-state error offers little benefit on a short indoor track and risks a slow, hard-to-diagnose overcorrection if left enabled.

### System Initialisation and the Startup Homing Routine

The `setup()` routine configures every pin's direction and then immediately performs the mechanical homing sequence described in Section 2.9.6, guaranteeing the lift carriage always starts a run from a known, repeatable reference position.

```
void setup()
{
  Serial.begin(115200);
  pinMode(LIMIT_SWITCH_PIN,INPUT_PULLUP);
  pinMode(L_RPWM,OUTPUT); pinMode(L_LPWM,OUTPUT);
  pinMode(R_RPWM,OUTPUT); pinMode(R_LPWM,OUTPUT);
  pinMode(BUZZER , OUTPUT);
  for(int i=0;i<8;i++) pinMode(sensor_pins[i],INPUT);

  stepper.setMaxSpeed(3000);
  stepper.setAcceleration(2000);

  // --- STARTUP HOMING ROUTINE ---
  stepper.setSpeed(stepperSpeed);
  while(digitalRead(LIMIT_SWITCH_PIN) == HIGH)
  {
    stepper.runSpeed();
  }
  stepper.setSpeed(0);
  delay(1000);
}
```

The limit switch is wired using the Nano's internal pull-up resistor (`INPUT_PULLUP`), so it reads logic HIGH while un-pressed and is pulled to LOW the instant it is physically triggered by the descending carriage. The homing `while` loop is therefore a plain-English statement of intent: *keep driving the carriage downward for as long as the switch has not yet been triggered.* The moment the switch closes, the loop exits, the stepper speed is forced to zero, and a one-second settling delay allows any residual mechanical vibration to die out before the main control loop begins.

### The Main Control Loop

The main loop's very first action, on every single pass, is to check for an obstacle – this check is given strict priority over line-following, station detec-

tion, and everything else, since a collision is a far more serious failure than a momentarily interrupted delivery.

```
void loop()
{
  if (checkObstacles()) { return; }

  float sum_norm=0, sum_norm_weight=0;
  for(int i=0;i<8;i++)
  {
    sensor_values[i]=analogRead(sensor_pins[i]);
    int val=constrain(sensor_values[i],white_value,black_value[i]);
    normalized[i]=(float)(val-white_value)/(black_value[i]-white_value);
    sum_norm+=normalized[i];
    sum_norm_weight+=normalized[i]*weight[i];
  }
  ...
```

If `checkObstacles()` returns `true`, the function returns immediately – the rest of the loop, including the IR read and the PID law, is simply skipped for that pass, and control returns to the very top of `loop()` to re-check for obstacles again on the next pass. Only once the path ahead is confirmed clear does the firmware proceed to sample and normalise all eight IR channels, using exactly the same clamp-and-normalise pipeline validated in Phase 1: each raw ADC reading is clamped between that channel's calibrated `white_value` and `black_value[i]`, then rescaled onto a clean $[0.0, 1.0]$ range.

### Station Arbitration Logic

With the normalised sensor data available, the firmware next decides which of four possible situations it is in – a pick-up station, a drop-off station, an actively tracked line, or a genuinely lost line – based purely on the aggregate value `sum_norm`.

```
  // PICKUP STATION (ALL BLACK)
  if(sum_norm>6.5)
  {
    white_count = 0;
    if(!box_loaded)
    {
      setMotorSpeeds(0,0);
      runPickupCycle();
```

```
    box_loaded=true;
    setMotorSpeeds(base_speed, base_speed);
    delay(500);
  }
  return;
}

// DROP STATION (ALL WHITE) / LOST LINE
else if(sum_norm<0.1)
{
  white_count++;
  setMotorSpeeds(0,0);
  if(white_count > 50)
  {
    if(box_loaded)
    {
      setMotorSpeeds(0,0);
      runDropCycle();
      box_loaded=false;
      setMotorSpeeds(base_speed, base_speed);
      delay(1000);
      setMotorSpeeds(0,0);
    }
    else { setMotorSpeeds(0,0); }
  }
  return;
}

// NORMAL LINE FOLLOWING
else if(sum_norm>0.4)
{
  white_count = 0;
  position=sum_norm_weight/sum_norm;
  if(position>2) last_side=1;
  else if(position<-2) last_side=-1;
}
else { white_count = 0; setMotorSpeeds(0,0); return; }
```

Because all eight sensors are individually normalised onto a [0,1] range, a fully saturated black surface pushes the sum of all eight channels toward its theoretical ceiling of 8.0; the threshold `sum_norm > 6.5` is set comfortably

below that ceiling to tolerate minor optical noise while still being far above anything a normal, narrow tracking line could ever produce, which is what makes the pick-up marker unambiguous to detect. The `box_loaded` boolean is the guard that prevents the vehicle from re-triggering a pickup cycle every single loop iteration while it happens to be sitting on the black tile – the cycle fires exactly once per approach, then the vehicle deliberately drives itself 500 ms forward off the marker before line-following logic resumes.

The drop-off branch reuses the same aggregate threshold logic from the opposite end of the scale, but adds the `white_count` debounce counter discussed in Section 2.9.7 before trusting the reading, and is itself gated by the same `box_loaded` flag so that an empty vehicle passing back over a drop station does nothing but stop safely. Only if neither special condition is met does the firmware fall through to ordinary proportional line-tracking, using the identical centre-of-mass weighted-average formula validated in Phase 1:

$$\text{Position} = \frac{\sum_{i=0}^{7}(N_i \cdot W_i)}{\sum_{i=0}^{7} N_i}$$

### Discrete PID Law and Motor Speed Computation

Once a valid `position` has been computed, the same discrete-time PID law introduced in Phase 1 converts it into a steering correction, which is then combined with the dynamic-braking logic that reduces cruising speed automatically on sharp turns:

```
error=position;
integral=constrain(integral+error,-50,50);
float derivative=error-last_error;
float correction=kp*error+ki*integral+kd*derivative;
last_error=error;

int dynamic_base=base_speed-abs(correction*0.8);
dynamic_base=max(dynamic_base,40);
int left_speed=dynamic_base+correction;
int right_speed=dynamic_base-correction;
left_speed=constrain(left_speed,-255,255);
right_speed=constrain(right_speed,-255,255);
setMotorSpeeds(left_speed,right_speed);
delay(5);
```

The continuous-time control law being approximated here is unchanged

from Phase 1:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}$$

with the discrete integral realised as a running accumulation clamped to $\pm 50$ to guard against integral windup, and the discrete derivative realised as a simple one-step backward difference against `last_error`. The only numerical change of substance versus Phase 1 is the dynamic-braking coefficient, which trims the cruising base speed more aggressively (`correction*0.8` rather than the earlier prototype's `correction*0.5`) in proportion to how hard the vehicle is currently steering – an appropriate adjustment given the final prototype's considerably larger turning radius and its greater sensitivity to skidding a loaded payload off the deck mid-turn.

### Obstacle Detection Routine

```
bool checkObstacles()
{
  int distL = sonarLeft.ping_cm();
  int distM = sonarMiddle.ping_cm();
  int distR = sonarRight.ping_cm();

  bool objL = (distL > 0 && distL <= STOP_DISTANCE);
  bool objM = (distM > 0 && distM <= STOP_DISTANCE);
  bool objR = (distR > 0 && distR <= STOP_DISTANCE);

  if (objL || objM || objR) {
    setMotorSpeeds(0, 0);
    digitalWrite(BUZZER, HIGH);
    delay(1000);
    digitalWrite(BUZZER, LOW);
    return true;
  }
  return false;
}
```

Each of the three `ping_cm()` calls returns `0` if no echo was received within the 50 cm ranging window configured for `MAX_DISTANCE` – a NewPing convention that must be explicitly checked for (`distL > 0`), since a raw zero reading would otherwise be misread as "an object zero centimetres away" rather than its true meaning of "nothing detected in range." If any one of

the three sensors reports a valid echo within the 15 cm `STOP_DISTANCE`, the drivetrain is halted immediately, the buzzer is driven solid HIGH as a continuous audible warning, and the firmware deliberately blocks for a full second before re-checking. This blocking `delay(1000)` is a conscious simplification appropriate to an educational platform – a production-grade AGV would instead implement this as a non-blocking, `millis()`-timed state so the vehicle could re-evaluate its sensors continuously during the pause rather than freezing entirely, but the simpler blocking approach keeps the safety-critical code path short, easy to read, and easy for a student to verify by inspection.

### Pickup and Drop Actuation Cycles

The two functions below drive the physical lift mechanism and are, together with the station-arbitration logic already discussed, the functional addition that distinguishes Phase 3 from both earlier prototypes.

```
void runPickupCycle()
{
  stepper.setSpeed(stepperSpeed);
  while(digitalRead(LIMIT_SWITCH_PIN)==HIGH) { stepper.runSpeed(); }
  stepper.setSpeed(0);
  delay(200);

  stepper.setSpeed(-stepperSpeed);
  unsigned long startMillis=millis();
  while(millis()-startMillis<13000)
  {
    stepper.runSpeed();
    buzz();
  }
  stepper.setSpeed(0);
  digitalWrite(BUZZER, LOW);
}

void runDropCycle()
{
  stepper.setSpeed(stepperSpeed);
  while(digitalRead(LIMIT_SWITCH_PIN)==HIGH)
  {
    stepper.runSpeed();
    buzz();
```

```
  }
  stepper.setSpeed(0);
  digitalWrite(BUZZER, LOW);
  delay(200);
}
```

`runPickupCycle()` begins with a fresh re-homing pass – exactly the same limit-switch-seeking loop used at start-up – to guarantee the carriage is at a known, repeatable position immediately before every single lift, regardless of any accumulated drift from the previous cycle. Only once home is re-confirmed does the stepper reverse direction and drive upward for a fixed duration of **13 seconds**, lifting the carriage (and the payload resting on it) clear of the pick-up station. This is a deliberately *open-loop, time-based* lift: because the mechanism has no linear encoder or top-limit switch, the firmware trusts that 13 seconds of constant-speed stepping reliably corresponds to the same physical lift height every time, provided the battery voltage (and therefore the stepper's available torque) stays reasonably consistent between cycles. This is an accepted engineering simplification for a low-cost educational build, and one of the clearest opportunities identified for future refinement (Section 2.9.14). `runDropCycle()` is the direct mirror image – it simply drives the stepper back down until the limit switch is triggered again, which is inherently self-correcting regardless of exactly how far the carriage travelled during the corresponding lift.

Both cycles call `buzz()` continuously for their full duration, giving a clear, ongoing audible signal that the vehicle is mid-cycle – distinct from the solid, unbroken tone used by `checkObstacles()` to flag a safety stop:

```
void buzz()
{
  if (millis() - lastToggle >= 500)
  {
    buzzerState = !buzzerState;
    digitalWrite(BUZZER, buzzerState);
    lastToggle = millis();
  }
}
```

This is a simple non-blocking toggle: rather than calling `delay()` (which would freeze the stepper's step pulses mid-lift), `buzz()` checks the elapsed time against `lastToggle` on every call and only flips the buzzer's state roughly twice a second, giving a slow, intermittent "beep – beep – beep"

pattern that is clearly distinguishable from the continuous obstacle-warning tone, all without ever blocking the tight stepper-pulsing loop it is called from.

#### Motor Actuation Abstraction Layer

Finally, `setMotorSpeeds()` translates a signed speed value for each side of the drivetrain into the correct `RPWM/LPWM` pair for its BTS7960 module:

```
void setMotorSpeeds(int left,int right)
{
  if(left>=0) { analogWrite(L_RPWM,left);  analogWrite(L_LPWM,0); }
  else        { analogWrite(L_RPWM,0);     analogWrite(L_LPWM,-left); }

  if(right>=0){ analogWrite(R_RPWM,right); analogWrite(R_LPWM,0); }
  else        { analogWrite(R_RPWM,0);     analogWrite(R_LPWM,-right); }
}
```

Because `analogWrite()` cannot accept a negative duty cycle, the function first tests the sign of the requested speed and then writes the corresponding *magnitude* to whichever of the two PWM inputs produces that direction of rotation on the BTS7960, holding the other input at zero. This single, compact abstraction is what allows the PID loop, the pickup/drop driving manoeuvres, and the obstacle-triggered emergency stop to all issue commands through one uniform interface, regardless of which higher-level behaviour is currently in control of the drivetrain.

### 2.9.11 Payload Handling and Structural Considerations

The final prototype is rated for a maximum payload of **1 kg**, a figure set by the combined mechanical margin of the lead-screw/lead-nut assembly, the NEMA 17's available holding torque at the operating current supplied by its driver, and the 3D-printed carriage plate's own structural stiffness under a distributed load. This is comfortably within the torque budget of the drivetrain itself – the dual Johnson motors, individually rated well beyond 1 kg of towing capacity, are never remotely close to their limit during normal operation – so the lift mechanism, rather than the drivetrain, is the deliberate limiting factor on total payload, which is the appropriate engineering choice for a platform whose primary purpose is demonstrating the pick-and-place mechanism itself.

### 2.9.12 Bill of Materials

Table 2.1 summarises the principal hardware used in the final prototype.

Table 2.1: Phase 3 Final Prototype – Bill of Materials

| Category | Component Description | Q |
|---|---|---|
| **Structure** | 2020 Aluminium T-Slot Extrusion (500×500 mm frame) | |
| | 90-Degree Corner Brackets, M4 T-Nuts & Screws | |
| **Propulsion** | 12 V Johnson-type DC Geared Motor (high-torque, ∼10 kg rated) | |
| | Large Passive Caster Wheel | |
| **Motor Drivers** | BTS7960 High-Current H-Bridge Module | |
| **Lift Mechanism** | NEMA 17 Hybrid Stepper Motor | |
| | Lead Screw & Matching Lead Nut | |
| | 3D-Printed Carriage, Guide-Rod Mounts & Couplers | |
| **Sensing** | 8-Channel Analog IR Reflectance Array | |
| | Ultrasonic Distance Sensor (obstacle avoidance) | |
| **Control & Feedback** | Arduino Nano (ATmega328P) | |
| | Mechanical Limit Switch / Passive Buzzer | |
| **Power** | 12 V, 2000 mAh, 35C LiPo Battery Pack | |

### 2.9.13 Testing and Validation

The completed prototype was validated against the same three functional requirements it was designed around. Repeated runs confirmed reliable line-tracking at the tuned $K_p = 10.0$, $K_i = 0.0$, $K_d = 35.0$ gains across straight sections and moderate curves, with the dynamic-braking term visibly slowing the vehicle through sharper corners rather than skidding wide. Station detection was tested by placing a dedicated black tile at a pick-up point and a plain white section at a drop-off point; the vehicle consistently halted and correctly initiated the lift or lower cycle only once at each marker, confirming that the `box_loaded` guard and the `white_count` debounce counter were both functioning as intended and preventing repeated re-triggering. The lift mechanism was tested against its rated 1 kg payload, successfully raising and lowering a boxed test load at both stations across repeated cycles, as captured during a live demonstration run (Figure 2.7). Obstacle avoidance was verified by manually placing an object in the vehicle's path at varying positions across its three ultrasonic sensors; in every trial the vehicle halted and sounded the buzzer before contact, correctly resuming line-following once the obstruction was removed.

### 2.9.14 Limitations and Future Scope

Several deliberate engineering simplifications in this build have been noted through this section, and are recorded here together as the clearest set of directions for future refinement of the platform:

- **Open-loop, time-based lift height:** The fixed 13-second lift duration assumes consistent stepper torque and battery voltage between cycles. Adding a second limit switch (or an optical/magnetic encoder) at the top of the lift's travel would make the pickup height self-correcting in exactly the same way the drop height already is.
- **Blocking obstacle-stop delay:** The 1-second `delay()` inside `checkObstacles()` freezes the entire firmware, including the stepper pulse train, for its full duration. Migrating this routine to a non-blocking `millis()`-based state machine would allow the vehicle to keep monitoring its sensors continuously during an obstacle pause.
- **Fixed base speed regardless of payload state:** The current firmware drives at the same `base_speed` whether or not `box_loaded` is true. A natural extension would be to reduce cruising speed automatically while carrying a payload, trading a small amount of cycle time for an additional margin of stability.
- **Single-pin ultrasonic wiring:** While pin-efficient, sharing a single I/O pin between trigger and echo on each ultrasonic sensor is more susceptible to cross-talk between adjacent sensors firing in quick succession than a conventional dual-pin wiring scheme; staggering the three `ping_cm()` calls with a short settling delay, or switching to dedicated trigger/echo pins, would improve ranging reliability.

None of these represent a failure of the current build – each is a conscious, cost-and-complexity trade-off appropriate to a first full-scale, educationally-oriented prototype – but they represent the natural next iteration of the platform should it progress beyond this internship.

**Conclusion of Phase 3:** The final prototype successfully closes both gaps identified at the end of Phase 2. Its rebuilt aluminium-extrusion chassis holds the IR array's calibration stable under a real, distributed payload, and its stepper-driven lead-screw lift gives the vehicle a genuine, automated pick-and-place capability rather than requiring a human to load and unload every trip. Combined with its triple-ultrasonic obstacle layer and its considerably re-tuned PID control law, the final build stands as a complete, self-contained

demonstration of an industrial-style unit-load AGV, built entirely from low-cost, widely available components suited to an educational setting.

## 2.10 Conclusion of the AGV Development Programme

Across all three phases, this project traced a deliberate arc from a minimal sensing-and-control proof-of-concept, through a load-bearing but still manually-serviced transport vehicle, to a fully autonomous pick-and-place unit-load AGV. The Phase 1 breadboard rig established that an 8-channel analog IR array and a discrete-time PID law, running entirely on an 8-bit Arduino Nano, could deliver smooth, stable line-tracking together with a reliable line-recovery behaviour. Phase 2 proved that the same electronics architecture could be re-housed inside a genuinely load-bearing chassis, while exposing the structural and functional limits – flexible HDPE walls and a lack of automated loading – that would define the requirements for the final build. Phase 3 then addressed both of those limits directly: a rigid, 500 mm aluminium-extrusion chassis; a high-current, BTS7960-driven propulsion system sized for the vehicle's much greater mass; and, most significantly, an automated NEMA 17 lead-screw pick-and-place mechanism that lets the vehicle load and unload a 1 kg payload entirely on its own, gated by nothing more than the same line-tracking IR array already present for navigation.

Taken together, the three prototypes demonstrate not only the layered engineering process of iterative failure analysis and redesign, but also the project's consistent underlying commitment to educational accessibility: every hardware decision, across every phase, was made in favour of an inexpensive, hobbyist-accessible component, so that the resulting AGV platform – and the full working firmware documented in this chapter – remains straightforward for another student to reproduce, study, and extend.

# Chapter 3

# Internship Outcomes and Conclusion

## 3.1 Comprehensive Learning Outcomes

The FOSSEE Semester-Long Internship at the Indian Institute of Technology (IIT) Bombay provided an intensive, hands-on proving ground that bridged the gap between theoretical academic concepts and applied industrial mechatronics. The iterative development of the three AGV prototypes documented in this report yielded technical and methodological learning outcomes across several engineering disciplines:

### 3.1.1 Embedded Systems and Firmware Engineering

- **Microcontroller Architecture Management:** Developed a rigorous understanding of 8-bit AVR microcontroller constraints (specifically the ATmega328P), including optimising C++ firmware to run continuous floating-point PID calculus, stepper pulse generation, and multi-sensor polling within the chip's tightly limited 2 KB of Static RAM.
- **Hardware Abstraction and Signal Processing:** Gained practical experience translating raw physical signals into actionable digital logic – continuous ADC normalisation of the IR array, generation of high-frequency PWM for both DC motor and BTS7960 control, and correct interpretation of ultrasonic time-of-flight measurements.
- **Library-Based Actuator Control:** Learned to integrate and correctly configure third-party Arduino libraries (`AccelStepper`, `NewPing`)

rather than re-implementing low-level timing logic by hand, and to reason about the trade-offs of blocking versus non-blocking control routines in a single-threaded firmware environment.

### 3.1.2 Applied Control Theory and Kinematics

- **Discrete-Time PID Implementation and Re-Tuning:** Successfully translated a continuous-time control law into discrete-time firmware, and – critically – learned to empirically re-tune its gains ($K_p$, $K_i$, $K_d$) as the physical system itself changed mass and inertia across the three prototypes.
- **Differential Drive Kinematics:** Gained practical expertise translating an abstract steering correction into independent left/right wheel velocities, including dynamic-braking logic to prevent a loaded vehicle from skidding through sharp turns.
- **Open-Loop Linear Actuation:** Learned the practical trade-offs of time-based, open-loop stepper positioning versus closed-loop position feedback, and how limit-switch homing can be used to bound the accumulated error of an otherwise open-loop actuator.

### 3.1.3 Mechanical Design and Iterative Prototyping

- **Structural Rigidity and Material Selection:** Learned the critical importance of chassis material choice through direct, hands-on failure analysis – from a flat acrylic prototype, through a flexible HDPE storage-bin chassis, to a rigid 2020 aluminium T-slot extrusion frame capable of holding sensor calibration stable under a real payload.
- **Rapid Manufacturing and CAD:** Built proficiency in CAD design and FDM 3D printing to engineer custom mechanical interfaces, including the lead-screw carriage and guide-rod mounts of the final lift mechanism.

### 3.1.4 Power Electronics and System Integration

- **High-Current Motor Driver Selection:** Learned to identify and correct a genuine electrical bottleneck – replacing a linear L298N H-bridge with switching BTS7960 modules once the drivetrain's current requirements outgrew the L298N's practical limits.

- **Noise-Aware Power Architecture:** Gained direct experience isolating a noise-sensitive analog sensing rail from a high-current PWM-switched motor rail using a deliberately partitioned dual-battery power architecture.

### 3.1.5 The Open-Source Engineering Paradigm

Throughout the internship, the FOSSEE methodology of using open-source hardware platforms (Arduino) and freely available software libraries to rapidly prototype genuinely functional, industrial-style solutions was thoroughly internalised. This approach dramatically reduced development cost and iteration time while never compromising the reliability of the finished platform – and, in keeping with the project's own educational objectives, ensures that the resulting AGV can be freely reproduced, studied, and extended by future students.

## 3.2 Conclusion

The primary objective of this internship was to explore, design, and empirically validate the core mechatronic components of a low-cost, educational Automated Guided Vehicle, and to progressively mature that platform from a bare sensing-and-control proof-of-concept into a genuinely autonomous, payload-handling industrial demonstrator. By taking the AGV through three deliberate design iterations – each one targeting a specific, previously-identified limitation of the prototype before it – this objective was conclusively achieved.

This iterative development cycle explicitly demonstrated the unforgiving dependencies between mechanical integrity, electrical power distribution, and algorithmic intelligence. The structural flex of the Phase 2 HDPE chassis, and the current limitations of the Phase 1/2 L298N driver once the vehicle scaled up in Phase 3, were invaluable, hands-on engineering lessons that textbook theory alone cannot adequately convey.

Ultimately, the rigorous, iterative design methodology championed by the FOSSEE project has profoundly enhanced my practical engineering capabilities as a Computer Engineering student. The final AGV platform engineered during this internship stands as a robust, reproducible, and genuinely autonomous pick-and-place prototype, successfully demonstrating the foundational principles of Industry 4.0 material handling on an entirely open, hobbyist-accessible hardware stack – and setting a firm intellectual foundation for future academic research in autonomous robotics and cyber-physical

systems.

# Bibliography


[1] Microchip Technology Inc., *ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash Datasheet*, Rev. A. Chandler, AZ: Microchip Technology, 2018.

[2] STMicroelectronics, *L298 Dual Full-Bridge Driver Datasheet*, Rev. 6. Geneva, Switzerland: STMicroelectronics, 2000.

[3] Infineon Technologies, *BTS7960B High Current PN Half Bridge Datasheet*, Rev. 1.1. Neubiberg, Germany: Infineon Technologies AG, 2008.

[4] Vishay Semiconductors, *TCRT5000 / TCRT5000L Reflective Optical Sensor with Transistor Output Datasheet*, Rev. 1.8. Malvern, PA: Vishay Intertechnology, 2009.

[5] ITead Studio, *HC-SR04 Ultrasonic Ranging Module User Guide*. Shenzhen, China: ITead Studio, 2013.

[6] Oriental Motor Co., Ltd., *NEMA 17 Hybrid Stepping Motor Specifications and Performance Data*. Tokyo, Japan: Oriental Motor Co., 2020.

[7] Pololu Corporation, *Micro Metal Gearmotor Specifications and Performance Data*. Las Vegas, NV: Pololu Robotics and Electronics, 2021.

[8] Panasonic Corporation, *NCR18650B Rechargeable Lithium-Ion Battery Product Specifications*. Osaka, Japan: Panasonic Automotive & Industrial Systems Company, 2012.

[9] M. McCauley, *AccelStepper: Arduino Library for Stepper and DC Motor Control*. Airspayce Pty Ltd, 2021.

[10] T. Eckel, *NewPing Library for Arduino: Ultrasonic Sensor Library*, v1.9. 2016.

[11] K. Ogata, *Modern Control Engineering*, 5th ed. Upper Saddle River, NJ: Prentice Hall, 2010.

[12] A. Kumar and S. Singh, "Design and Development of Automated Guided Vehicle with Line Follower Concept using IR," *2023 Fifth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, Veladas, India, 2023, pp. 1-6.

[13] Z. Wang, "OpenCV-based PID Control Line Following Vehicle with Object Recognition and Reaction," *AIP Conference Proceedings*, vol. 3085, no. 1, 2024.

[14] M. Hermann, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios," *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Koloa, HI, USA, 2016, pp. 3928-3937.