DandeBot - An Autonomous Weeding Solution for Residential Lawns

by

Nishanth Rajkumar

A Thesis in the Department of
Mechanical, Industrial and Aerospace Engineering
Presented in Partial Fulfillment of the Requirements
for the Degree of
Master of Applied Sciences in Mechanical Engineering
at Concordia University
Montreal, Quebec, Canada

July 2025

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared by: **Nishanth Rajkumar**

Entitled: **DandeBot - An Autonomous Weeding Solution for Residential Lawns** and submitted in partial fulfillment of the requirements for the degree of: **Master of Applied Sciences in Mechanical Engineering** Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. Brandon Gordon

_____ Examiner
Dr. Brandon Gordon

_____ Examiner
Dr. Susan Liscouet-Hanke

_____ Thesis Supervisor
Dr. Wen-Fang Xie

Approved by:

_____
Dr. Sivakumar Narayanswamy
Chair of Department or Graduate Program Director

_____
Dr. Mourad Debbabi
Dean, Gina Cody School of Engineering and Computer Science

Abstract

DandeBot – An Autonomous Weeding Solution for Residential Lawns
Author: Nishanth Rajkumar

This thesis presents the development and validation of DandeBot, an autonomous robotic system designed for comprehensive residential lawn maintenance. The robot addresses the need for efficient, eco-conscious, and low-maintenance lawn care through a fully electric platform powered by an AI-driven software stack. Emphasizing safety, adaptability, and ease of use, the hardware was developed using CAD and Design for Manufacturing (DFM) principles, resulting in a modular and robust design. The integrated software stack combines localization, mapping, and path planning using odometry, visual odometry, and IMU data fusion to navigate dynamic outdoor environments.

Task-specific algorithms were developed and validated for autonomous navigation, weed detection, and obstacle avoidance. Key hardware innovations include a modular gripper system for weed removal and adaptable attachments for multiple lawn care tasks. Field trials confirmed the robot's capability to perform with high precision and reliability in varied lawn conditions, significantly reducing the need for human intervention.

This work contributes to the growing field of service robotics by demonstrating how intelligent systems can automate routine household maintenance. The thesis concludes by outlining future research directions, including system scalability, enhanced multi-tasking capabilities, and integration with smart home networks.

Acknowledgements

Foremost, I would like to express my sincere gratitude to all of my supervisors for their unwavering support, invaluable insights, and expert guidance throughout the duration of this project. Their deep knowledge and relentless encouragement have been instrumental in navigating the complexities of this work.

I owe a profound debt of gratitude to my family, whose enduring belief in my abilities and steadfast encouragement have been my pillars of strength. Their constant reassurance and support have been paramount in realizing this project.

To my friends, thank you for the countless brainstorming sessions, the constructive criticism, and the camaraderie. Your perspectives and feedback have enriched this project in ways beyond measure.

Lastly, I'd like to extend my appreciation to various professional forums whose knowledge-sharing and collaborative spirit provided crucial insights and resources. Their dedication to advancing the field and fostering a spirit of collaboration has been instrumental in the successful completion of this project.

Due to the support I received, I was also able to secure multiple awards, among which the GCS Innovation Funds was a notable recognition.

To all mentioned and those unmentioned who have, in one way or another, contributed to this journey, I express my deepest thanks. Your collective influence has left an indelible mark on this project and my professional journey.

Contents

List of Tables

List of Figures

Listings

Nomenclature

α	Angle to the goal from the robot's current orientation
β	Angle between robot's heading and the goal's orientation
ω	Robot's angular velocity
ω_l, ω_r	Left and right wheel angular velocities
ρ	Distance between robot and target (in polar coordinates)
θ	Robot's orientation with respect to X-axis
d	Distance between the wheels (wheelbase)
ICC	Instantaneous Center of Curvature
R	Radius of curvature of the robot's path
r	Radius of each wheel
v	Linear velocity of the robot
V_l, V_r	Left and right wheel linear velocities
x, y	Robot's position coordinates in global frame
AI	Artificial Intelligence
CAN	Controller Area Network
CNN	Convolutional Neural Network
DC	Direct Current
DFM	Design for manufacturing
DWB	Dynamic Window Approach
GPS	Global Positioning System

IMU Inertial Measurement Unit

LIDAR Light Detection and Ranging

MCU Microcontroller Unit

ODE Ordinary Differential Equation

PD Proportional-Derivative controller

PID Proportional-Integral-Derivative controller

PWM Pulse Width Modulation

ROS Robot Operating System

RTK Real-Time Kinematic

SBC Single-Board Computer

SoM System on Module

SSH Secure Shell

UART Universal Asynchronous Receiver Transmitter

UGV Unmanned Ground Vehicle

YOLO You Only Look Once

Chapter 1

Introduction

In Canada, maintaining pristine lawns during spring and summer is a challenge due to the widespread presence of dandelions and many other types of weeds. Addressing these issues, a state-of-the-art robotic solution needs to be developed that autonomously detects and eradicates these plants, using advanced AI algorithms and cutting-edge design, thus revolutionizing lawn maintenance and eliminating manual intervention.

1.1 Project Motivation and Objectives

Lawn maintenance, particularly weed control, has historically posed a unique set of challenges for gardeners and homeowners alike. In the vast landscapes of Canada, the presence of dandelions during springs and summers has been a consistent concern. Although these flora can be admired for their aesthetic value in certain contexts, in a well-manicured lawn, they often signify neglect and can be detrimental to the health of the desired grasses and plants. Consequently, the persistent and rapid growth of these resilient weeds requires a significant amount of manual labor and careful attention.

Over the years, various solutions have been proposed and adopted. These ranged from manual removal tools to chemical herbicides. Several mechanical devices designed to grasp or hold objects were developed to aid in the removal process. On the other end of the spectrum, chemical herbicides targeting weeds presented a more hands-off solution, though they brought along their own set

of environmental concerns. However, these methods either lacked the required efficiency or presented sustainability and environmental challenges.

With advancements in technology and the increasing demand for sustainable and efficient solutions, the focus has shifted towards automation. The realm of robotics and automation has shown promise in numerous fields, from industrial manufacturing to household chores. The increasing accessibility and advances in sensors, actuators, and computing power have made it possible to consider automated solutions even for challenges as complex as lawn maintenance.

Given this context, the idea of a mobile robot capable of autonomously navigating a garden, identifying, and eradicating weeds becomes an attractive proposition. Not only does it eliminate the need for manual intervention, but, with the right design and implementation, it can also be environmentally friendly and highly efficient. This project explores the potential of such a solution, combining state-of-the-art technologies in robotics, computer-aided design, and advanced algorithms, to address the long-standing challenge of lawn maintenance in Canada.

The thesis aims to bridge the gap between traditional gardening methods and advanced robotics, offering a practical solution to a common household problem. By developing and validating this autonomous system, this work demonstrates how robotics can transform routine lawn care into a more sustainable and accessible process.

1.2 Contribution of Project

This project presents the design and implementation of an autonomous lawn care robot using the ROS2 Iron framework. The robot was developed to handle comprehensive lawn maintenance tasks such as mapping, systematic area coverage, weed detection and extraction, mowing, and health monitoring. The key contributions of the project are as follows:

- **Custom-Built Differential Drive Robot:** A complete outdoor robot platform was built from scratch, featuring a differential drive chassis equipped with a 2D LIDAR, a vision camera, and an IMU, optimized for performance in uneven lawn environments.

- **Remote Teleoperation System:** A ROS2-based teleoperation interface was implemented to enable manual control during setup, testing, and exception handling, providing visual feedback and real-time robot control.
- **Sensor Fusion with EKF:** Odometry from wheel encoders and orientation data from the IMU were fused using an Extended Kalman Filter (EKF) via the `robot_localization` package to deliver accurate and drift-reduced pose estimation for reliable mapping and navigation.
- **Real-Time SLAM for Lawn Mapping:** SLAM Toolbox was utilized to perform 2D LIDAR-based SLAM, incrementally generating a live occupancy grid map. The system incorporated loop closure to maintain consistency and correct drift as the robot navigated the lawn.
- **Custom Coverage Path Planner with Dynamic Obstacle Avoidance:** A complete coverage path planner was built on top of the Fields2Cover library to divide the lawn into efficient mowing strips. The default Nav2 behavior tree was modified to incorporate waypoint-based navigation with real-time dynamic obstacle detection and avoidance, allowing the robot to adaptively skip or revisit areas based on environmental changes.
- **Integrated Weed Detection and P2P Extraction:** A YOLOv8-based computer vision model, trained and deployed by teammate Ibrahim Babikar, enabled real-time weed detection. Detected weeds were mapped with coordinate references, and a point-to-point navigation routine was executed to reach and remove each weed using a custom mechanical extractor inspired by the Fiskars weeder [1].
- **Automated Mowing System:** The mowing blade system was integrated into the navigation behavior tree, activating only during planned coverage operations. The robot followed optimized coverage paths to ensure uniform lawn height with minimal overlaps and missed zones.

These contributions together demonstrate a robust, AI-enhanced, end-to-end robotic system for lawn maintenance. The robot combines autonomous navigation, real-time perception, and task execution in a modular, scalable framework built entirely with open-source tools and custom-designed hardware.

This thesis validates the system's performance through controlled field testing, showing that the robot effectively addresses real-world challenges such as weed management, dynamic environments, and terrain variation. By bridging traditional landscaping methods with intelligent automation, the project illustrates how robotics can turn labor-intensive outdoor work into a safe, efficient, and sustainable process.

Future improvements may include smart home integration, enhanced multi-robot coordination, and large-scale deployment, paving the way for greater adoption of autonomous robotics in everyday residential settings.

1.3 Thesis Outline

This thesis presents the development of an autonomous robot designed to handle a full range of lawn maintenance tasks, including mowing and weeding with a special focus on addressing dandelion proliferation in residential gardens. The goal of this project is to integrate robotics, perception, and motion planning into a single system that provides an efficient, eco-friendly, and user-friendly solution to everyday lawn care problems.

The thesis is organized into several chapters, each focusing on a critical aspect of the robot's development:

Chapter 2: Literature Review primarily explores some of the typical problems faced by homeowners with lawn maintenance. Further, the review delves into some of the existing solutions, demonstrating the necessity for innovation in lawn care. With the current AI and Robotics hype, the mobile manipulators were examined and analyzed for our problem. Finally, a few patents and some existing resources and projects were used to study about the design principles involved.

Chapter 3: In this chapter, the key issues of locomotion were studied. Moreover, some case studies were deeply analyzed in different types of wheeled robots. Earlier prototypes were developed to validate each conceptual step of the final prototype carefully.

Chapter 4:All the intricate electronic components were discussed in this chapter including system on module, actuators and its drivers with a wide

variety of sensors. Furthermore, implementation of different communication protocols between these components was successfully carried out.

Chapter 5: This chapter covers the entire process from collecting and labeling data, training a neural network to recognize and classify it, running inference and tracking, and finally integrating the whole system into the controller.

Chapter 6: The control section of the thesis includes both low-level and high-level components. The high-level controller determines the linear and angular velocities, which are then converted into individual wheel velocities and passed to the low-level controller.

Chapter 7: This chapter explores experimental test and validation of the advanced robotic functionalities, including remote teleoperation, intelligent mapping, complete coverage and detection, point-to-point weed extraction, and mowing.

Chapter 8: The final chapter summarizes the work carried out in the project and discusses some of the work that could be done in the future. It highlights plans to integrate aerating and leaf sweeping, make refinements and software upgrades, and finalize the embodiment design for the minimum viable product.

1.4 Summary

An overview of the project has been discussed, beginning with the motivation for developing an autonomous solution to address common challenges in residential and commercial lawn maintenance. The primary objectives of the project were defined, focusing on enabling software for automated mowing, weed detection and extraction, efficient path coverage, and environmental monitoring using a ROS2-based robotic platform. A summary of the key contributions has also been outlined, which include the design and development of a custom-designed differential-drive robot, integration of advanced perception and navigation systems, implementation of a dynamic coverage planner, and deployment of AI-driven weed removal. Finally, the structure of the thesis has been introduced, providing a roadmap for the technical methods, implementation details, experimental results, and overall evaluation presented in the following chapters.

Chapter 2

Literature Review

In this literature review, a comprehensive analysis of the existing challenges in lawn maintenance including the time and cost spent with extensive breakdowns by lawn size, professional vs DIY maintenance, costs of mowing, weeding, and other tasks over time is conducted. Furthermore, the value and adoption of robotic lawn solutions, with a full cost-benefit breakdown, tool costs, and usage trends are evaluated.

Beginning with an exploration of the environmental and economic issues associated with traditional lawn care, the review delves into the necessity for more sustainable and efficient solutions. It examines the evolution of mobile autonomous systems and their configurations, highlighting their potential for revolutionizing residential lawn care. Core features of autonomous weeding solutions, such as precision, safety, and resource efficiency, are explored in detail, along with the broader implications of integrating all-in-one solutions for lawn maintenance. Additionally, considerations around sustainability, human-robot interaction, and regulatory standards are discussed to provide a holistic view of the field. This review aims to establish a foundation for the development of DandeBot, a fully autonomous and eco-friendly weeding and mowing solution, while outlining future possibilities for enhancing its functionality into a versatile lawn care system.

2.1 Problem Identification: Environmental and Ecological Impact

Intensive lawn care practices, such as excessive fertilization, irrigation, and mowing, are significant contributors to environmental challenges, including groundwater contamination, greenhouse gas emissions, and ecosystem disruption [2]. Improper management of lawns not only diminishes their ecological benefits but also worsen environmental problems. For instance, while healthy lawns have the potential to sequester carbon, excessive use of nitrogen-based fertilizers leads to nitrous oxide emissions, and a greenhouse gas that is far more potent than carbon dioxide [2]. Additionally, traditional lawn care equipment, which predominantly relies on fossil fuels, further increases urban carbon footprints, making residential lawns net carbon sources rather than carbon sinks [3].

Homeowner preferences, often driven by aesthetics and social norms, perpetuate high-input, low-diversity lawn systems. These systems require frequent mowing, watering, and weeding, which contribute to labor demands and environmental impacts, such as biodiversity loss and excessive water consumption [4]. Despite the cultural value of lawns as symbols of aesthetic and recreational spaces, their ecological costs are often overlooked. This oversight fosters a cycle of unsustainable practices that prioritize appearance over environmental health [5].

A significant knowledge gap among homeowners further exacerbates the issue. Many lack awareness of best management practices (BMPs), such as soil testing, the proper application of herbicides, and efficient water use [6]. This gap often leads to environmentally detrimental behaviors, including over watering, and improper mowing, which harm urban ecosystems. The absence of widespread education and awareness campaigns on sustainable lawn care practices results in limited adoption of eco-friendly strategies, compounding the environmental challenges.

Economic inefficiencies also play a role in the problem. The high costs associated with traditional lawn maintenance, including labor and resource consumption, often lead to neglected or poorly maintained lawns [7]. This neglect reduces the ecological and aesthetic benefits of green spaces, further highlighting the

inefficiencies in current practices. The reliance on high-resource inputs like water, herbicides, and fossil fuels not only increases operational costs but also contributes to the degradation of natural resources, creating a pressing need for sustainable alternatives.

The environmental inefficiencies of traditional lawn care are underscored by their impact on urban greenhouse gas budgets. Soil respiration from lawns and mulched areas emits more carbon than is sequestered by urban vegetation, further highlighting the ecological imbalance caused by conventional practices [3]. Additionally, resource-intensive practices like frequent mowing and irrigation deplete natural resources while offering minimal ecological benefits.

Addressing these challenges requires a paradigm shift in how lawns are managed. Innovative solutions, such as autonomous systems like DandeBot, offer a promising alternative to traditional practices. By integrating advanced technologies such as sensor fusion, AI-powered weed detection, and precision mowing, these systems can minimize environmental harm while optimizing resource use. Autonomous solutions are designed to align with BMPs, reducing emissions, and conserving water. They also address labor demands by automating routine maintenance tasks, providing a sustainable, efficient, and eco-friendly alternative to manual lawn care.

2.2 Necessity for Innovation in Lawn Care

2.2.1 Lawn Care Time and Cost Analysis (2005-2025)

Overall Spending: Homeowners in the U.S and Canada have significantly increased their lawn care spending over the past two decades. In the United States, average annual household expenditures on lawn care (including DIY supplies and hired services) roughly doubled from the mid-2000s to the mid-2020s. For example, the average U.S. household spent only about $100 per year on professional gardening/lawn services in 2007, but this grew to about $202 by 2002 [8]. When including DIY purchases (mowers, fertilizer, seed, etc.), total lawn and garden spending per household hovered around $120 in the late 2010s, then surged by 29% to roughly $155 in 2022 amid pandemic stay-at-home trends [9]. Many households took up lawn care and gardening during COVID-19, driving up sales of supplies and equipment. In Canada, a

similar spike occurred - the average household's lawn and garden expenditures jumped from about $408 CAD in 2018 to $524 CAD in 2021 [10, 11]. This reflects heightened interest in year improvements during the pandemic. Over the 2005-2025 period, Canadian homeowners' lawn care spending rose overall (e.g. it averaged in the low $400s CAD in the 2010s and climbed above $500 CAD by the 2020s).

Time Investment: Despite spending more money, the time Americans devote to lawn care has seen a slight downward trend through the 2000s and 2010s (with a recent uptick during COVID-19). As shown in Figure 2.1, the U.S Bureau of Labor Statistics' American Time Use Survey shows that the share of Americans age 15+ who do yard work on any give day declined from around 10.3% in 2005-2007 to a low of 8.7% in 2019 [12]. This indicates fewer people doing regular yard chores as of late 2010s, possibly due to busier lifestyles and more outsourcing. However, interest rebounded in 2020-2021 when many stayed at home (gardening activity roughly doubled in 2020 vs 2019 [12]. On an individual level, the average American homeowner who maintains their own lawn spends about 7-8 house per year mowing (over 384 hours in a lifetime of mowing) [13]. The average masks big differences based one lawn size and region: in cities with small yards (e.g. Cambridge, MA), people spend under 1-2 hours/year mowing whereas in areas with large lawns (e.g. Augusta, GA) homeowners average 20-27 hours/year mowing [13]. Beyond mowing, additional time is spent seasonally on weeding, edging, raking leaves and fertilizing, which can add a few more hours per month in active lawn-growing seasons.

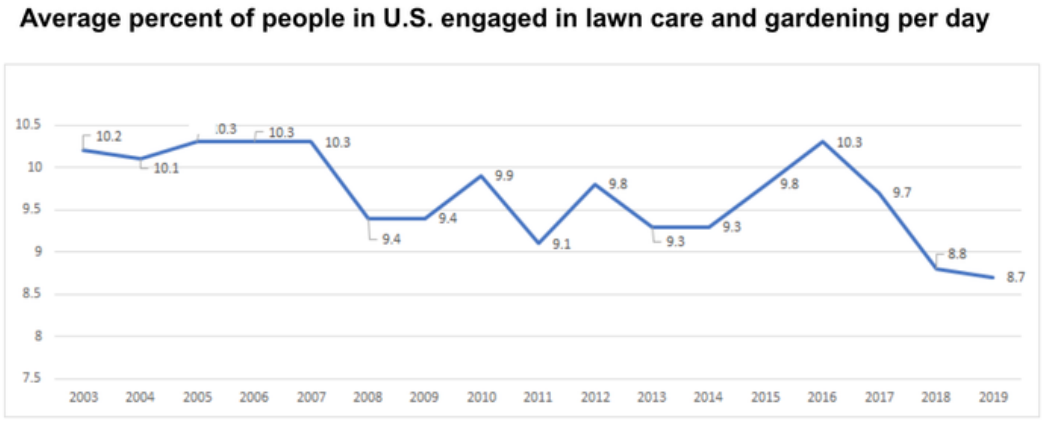

Figure 2.1: Average Percent of people in U.S. engaged in lawn care and gardening per day [12]

Residential Lawn Size and Cost Differences: The size of a lawn dramatically affects both the time and money required to maintain it. Small urban or suburban lawns (for instance, 1/8 acre or less) are relatively quick to mow and cheap to care for, whereas large properties (1/2-1 acre or more) demand far more hours and higher costs. A recent cost guide shows that hired maintenance services for a 1/4-acre lawn run around \$200-\$400 per month, while a 1-acre lawn can cost \$800-\$1600 per month for full service [14]. In contrast, a tiny 1/8-acre yard might only cost \$100-\$200 monthly with professional lawn care [14]. Table 2.1 summarizes typical service costs by lawn size (in the U.S.):

Table 2.1: Average monthly lawn maintenance costs by lawn size (Source: [14])

Lawn Size	**Average Monthly Maintenance Cost (Hired)**
1/8 acre (small)	\$100 – \$200
1/4 acre (medium)	\$200 – \$400
1/2 acre (large)	\$400 – \$800
1 acre (very large)	\$800 – \$1,600
2 acres (estate)	\$1,600 – \$3,200

The owners of large lawns not only face higher service fee if they outsource, but also spend more time if they DIY. As noted, respondents with bigger yards or extensive landscaping "**spend more time on yard work**" than those with

smaller yards [13]. Big lawns often require riding mowers or longer mowing sessions, more fertilizer, more weed control, and more irrigation, all of which increase both labor hours and expenses.

Hiring Services vs. DIY: A major decision for homeowners is whether to hire lawn care services or do it themselves. In the U.S., roughly 40% of homeowners hire professional lawn or landscaping help [12], and this rate has slowly increased as people become less inclined to do manual yard work. Higher-income households in particular are big consumers of professional lawn care (households earning > $100k are the top spenders in the lawn market) [15]. Hiring professionals saves personal time but can be expensive - a Harris Survey in 2002 found that households using professional lawn services spent about **$1200 per year** on those services [16], and that cost has likely risen with inflation (e.g. many might now spend $1500+ per year for regular service). By 2025, the **average U.S. homeowner** (across all lawn sizes) spends about $300/year on lawn care (many doing basic tasks themselves) [14], but those who fully outsource often pay much more (several hundred dollars per month as shown above. Homeowners who **DIY** their lawn care avoid labor fees but incur costs in equipment, fuel, and supplies. One analysis found that when you account for buying a mower, gas and maintenance, it could take about **7 years** of mowing to "break even" financially compared to hiring a mowing service [13]. In other words, initially, doing it yourself can be costly (a quality lawn mower, trimmer, etc. easily total $300 - $500+ upfront), but over the long run DIY can save money if one already owns the equipment. The trade-off, of course, is the **time** - many homeowners ultimately value the hours saved by outsourcing. Millenials moving into their first homes, in particular, show a tendency to **"outsource yard work"** because they are accustomed to on-demand services and want to spend weekends on other activities [12]. Nonetheless, a majority still do some or all of their lawn care themselves, often because they take pride in their lawn or believe it's cheaper.

Cost-Benefit Comparison: Whether a robotic mower is cost-effective depends on what you compare it against. If a homeowner is currently **hiring a lawn service** for, say, $50 per week, that's roughly $200 per month or $1000+ per season. In that scenario, a $1000 robot mower could **pay for itself in one year** by eliminating those service payments (aside from initial setup effort). Over several years, the electricity to charge the robot is only a few dollars

per month - far cheaper than paying a crew. This makes robots appealing for those with high ongoing maintenance fees. In fact, industry reports highlight "**operational cost efficiency** as a key benefit of robot mowers - they offer a **more cost effective approach** than a gasoline mowers by cutting out fuel and reducing labor needs [17] For homeowners who **do their own mowing**, the cost calculus is different: a $300 traditional mower vs a $1000+ robot. Here, the robot may eventually save **time** (which some may translate to money) and a but of ongoing cost (no gas, no personal labor). CNET notes **"you may ultimately save money over time"** with a robot, but only if you value your time or avoid future mower purchases [18]. It's often the time-saving convenience that justifies the robot for DIYers rather than pure dollar savings. Importantly, as more models have entered the market, prices are slowly trending down. By scaling up production, manufacturers have started to chip away at costs [18], so robotic mowers in 2025 are generally cheaper and more capable than those a decade prior.

2.2.2 Breakdown of Lawn Maintenance Tasks and Cost over Time

Traditional lawn care practices face a wide range of challenges that demand innovative solutions. Currently, manual labor and gasoline-powered mowers dominate the industry. These methods are not only labor-intensive and inefficient but also cause significant environmental damage. Gas-powered equipment is a major contributor to air pollution, greenhouse gas emissions, and noise pollution, worsening environmental issues in urban areas [19]. As cities grow and the global emphasis on resource efficiency increases, sustainable and automated alternatives in lawn care are becoming essential.

Mowing: Mowing the grass is the most routine lawn chore. In 2005, a typical homeowner with a medium lawn might mow about once a week in season, often using a gas-powered push mower. By 2025, this frequency remains common (weekly or bi-weekly mowing), but there's been a shift in equipment: **batter-electric mowers** have become popular alongside traditional gas mowers. This shift has been encouraged by environmental regulations - e.g. some jurisdictions in the U.S. are pushing landscaping businesses to use battery equipment over gas for noise and emission reasons [15]. Battery mowers in the 2020s offer

comparable power for typical lawns and avoid the ongoing cost of gasoline. The cost of mowing (if hiring someone) has risen with wages and fuel costs: landscapers charged around $50 - $100 per hour in the mid-2010s [15], and still do, with hourly rates in the $35 -$70+ range for basic mowing in 2025 [14]. Many services price mowing per visit or per square foot; for example, one source lists $50 - $205 as the range for a single lawn mowing service (the low end being a small yard, the high end being a very large yard) [14]. For DIY, fuel prices fluctuated over time (e.g. gas price spikes in 2008 and 2022 made mowing slightly more expensive those years), but overall the out-of-pocket cost to mow it yourself (fuel, maintenance) remains relatively low - on the order of a few dollars per mow for gas, or a bit of electricity for an electric mower. The bigger "cost" in DIY mowing is time, as discussed. Over 2005 - 2025, mowing technology improved (more efficient engines, cordless electric mowers, even **robotic mowers** discussed later) to reduce the burden on homeowners.

Weeding and Pest Control: Weed management, a key component of lawn care, also highlights the limitations of traditional methods. Manual and chemical weed control techniques are not only time-consuming but also environmentally harmful, often leading to soil degradation, water pollution, and loss of biodiversity [20]. The emergence of herbicide-resistant weeds has further emphasized the need for alternative approaches. Keeping weeds at bay is another constant task. Traditionally, homeowners spent time pulling weeds by hand or applied chemical weed killers (herbicides) periodically. In the mid-2000s, it was common to use "weed and feed" fertilizer products or spray weed killers on lawns; many lawn care companies offered weed control applications as part of their service packages. The cost for weed control services ranges widely depending on the extent of infestation - roughly $35 up to $400 per treatment in current estimates [14] One major change, especially in Canada, has been the **reduction of chemical pesticide use** for cosmetic lawn care. Many Canadian provinces enacted bans on certain lawn herbicides and pesticides around 2009 - 2010, shifting homeowners toward organic or manual weeding methods. In the U.S., while not banned outright (except in some municipalities), there is growing awareness of environmental impact. Still, Americans use a huge volume of lawn chemicals: roughly **78 million pounds of pesticides** (and **90 million pounds of fertilizer)** are applied to U.S. lawns each year to keep them green and weed-free [16]. Over time, the cost of herbicide products has increased slightly (and some

effective ones were phased out due to regulations), so homeowners today might spend a bit more on safer or specialty weed treatments. Weeding by hand costs only time, but can be labor-intensive for large areas - commercial properties and many homeowners thus opt for professional treatments periodically.

Fertilizing and Watering: Regular fertilization helps maintain a lush lawn. From 2005 to 2025, fertilizer prices saw some volatility (for instance, spikes in global fertilizer costs around 2008 and 2021. For consumers, a typical 5,000 sq.ft lawn might require a few bags of fertilizer per year, costing maybe $50 - $100 annually in the 2000s, rising to perhaps $100 - $150 by the 2020s due to price increases. Some homeowners have cut back on fertilizer use for environmental reasons, since excess runoff causes pollution. Many lawn care services include fertilization in a seasonal package - in 2025, the **cost to fertilize an average lawn** is around **$100 to $300 per application**, depending on lawn size and fertilizer type [14]. This is somewhat higher than in 2005, partly due to higher material costs and labor. Watering is another critical aspect: in dry regions or summer droughts, watering the lawn can be a time - consuming task (or a costly one if using automatic sprinklers). Water costs have risen, so efficient irrigation is a focus now. **Smart irrigation sensors** and weather-based controllers emerged in the 2010s, allowing automated sprinklers to adjust to rainfall and soil moisture. These sensors and controllers can save significant water (studies show anywhere from 6% up to 50+% water savings) and thus reduce water bulls for lawn maintenance [21]. Adoption of smart irrigation has grown on large commercial landscapes and upscale residences, aiming for cost savings and sustainability. overall, fertilizing and watering practices have trended toward less is more - using the right amount at the right time to avoid waste and extra cost.

Other Maintenance Tasks: Edging and trimming (cutting grass along driveways, flower beds and fences remains a regular task; strong trimmers ("weed-whackers") are commonly used. The equipment cost for trimmers has followed the mower trend - many electric battery models by 2025 instead of gas. **Aeration** (poking small holes in the lawn to alleviate soil compaction) is done occasionally (perhaps annually) by some homeowners or pros, costing about $75 - $200 per service in 2025 [14]. **Leaf raking/removal** in fall can consume many hours for those with leaf shedding trees; some pay for fall cleanup services (often $200 - $500 for a one-time yard cleanup for an average

property). Over the years, new tools like mulching mowers and leaf blowers (now also available in battery-powered versions) have made leaf management easier, though leaf blower noise and emissions have raised complaints, leading to some local bans on gas blowers. **Landscape renovation** or re-sodding the lawn is a rare, larger expense - in 2005 one could re-sod a lawn for maybe $1,000 (for a small yard), whereas in 2025 sod and labor might cost twice that. In general, the **cost of lawn care tools and supplied** has climbed modestly with inflation. U.S. sales of lawn equipment and garden supplied topped **$24 billion** annually by the mid-2010s [15], reflecting robust consumer spending on mowers, trimmers, spreaders, and the like. The industry has also seen a push towards **batter-powered** and even **"smart"** equipment (Bluetooth-enabled sprinkler controllers, robotics devices, etc.), giving consumers new options to maintain their lawns more efficiently if they are willing to invest upfront.

2.3 Robotic Lawn Care Solutions: Cost-Effectiveness and Adoption

The integration of mobile robotics into agriculture and lawn care has seen significant advancements over recent decades, transitioning from basic mechanization to highly sophisticated autonomous systems. Mobile robotics now play a pivotal role in addressing the growing demand for sustainable and efficient practices in these fields. Historical milestones, such as the introduction of GPS-based precision agriculture in the 1980s and the adoption of AI and machine learning in the 2010s, have laid the foundation for modern mobile autonomous systems capable of handling diverse tasks and challenging environments [22]. As shown in Figure 2.2, these advancements highlight the evolution of mobile robotics from basic mechanization to modern autonomous systems.

Figure 2.2: A visual representation of the evolution of mobile robotics from basic mechanization to modern autonomous systems [23, 24]

In 2020s, an array of **robotic lawn care tools is introduced** - most notably robotic lawn mowers - that promise to save time and labor. These autonomous mowers, which quietly trim the grass on a programmed schedule, have evolved from niche gadgets to a growing market segment. The question is whether these robotic solutions are cost-effective investments compared to traditional lawn care methods.

Figure 2.3: Robomow City 110 model

Figure 2.3 shows a robotic lawn mower in operation. Early robot mowers appeared in the late 1990s, and by the 2020s they have become more sophisticated and somewhat more affordable for consumers [18]. Modern units use sensors, GPS, or boundary wires to navigate yards and keep grass trimmed automatically.

Mobile autonomous systems are designed to perform complex operations such as planting, irrigation, weeding, harvesting, and pest control with high precision and minimal environmental impact. These systems utilize multi-sensor technologies, machine vision, and advanced control algorithms, enabling real-time monitoring and decision-making. By reducing reliance on labor-intensive manual practices, they offer improved efficiency, adaptability to varying terrains,

and optimized resource use. For instance, autonomous weeding robots minimize chemical application, while precision spraying robots enhance pesticide use, and contributing to more sustainable practices [25].

Adoption trends: Adoption of robotic mowers has been steady but gradual. Globally, the market for robotic mowers has grown at double-digit rates - valued around **$1.5 - $2.3 billion** in the early 2020s and projected to reach over **$4 billion by 2029** [26, 27]. In terms of units, an estimated **3.9 million** robotic mowers were in use worldwide in 2023, with forecasts of about **7 million units by 2029** [17]. **Europe** leads this trend (robot mowers are popular in countries like Germany and Sweden, where smaller lawns and higher labor costs make them attractive), while **North America** lags slightly but is catching up [17]. In the U.S. and Canada, robotic mowers are still views by many as a novel luxury - CNET notes that they "**still feel like an expensive novelty to many folks**" and adoption has not hit the inflection point that robotic vacuums did [18]. However, awareness and interest are rising as prices slowly come down and the technology proves itself. Early adopters (often tech enthusiasts or those with large lawns) have reported positive experiences, and even some commercial entities (like gold courses and city parks) are testing autonomous mowers to reduce labor needs. Overall, while only a small fraction of North American lawns are maintained by robots as of 2025, the trend is upward, especially in higher-income residential markets and for tech-savvy homeowners.

Costs of Robotic Tools: The **upfront cost** of a robotic mower is its biggest drawback. Consumer-grade robot mowers in 2025 range from around **$700 on the low end to $1200+ for mid range models** [18]. High-end models capable of handling very large or complex lawns (multiple acres, lots of obstacles) can cost $4000 - $6000 [18]. By comparison, a traditional walk-behind lawnmower might cost $250 - $500 (or as little as $100 for a basic model [18]). Thus the robot requires significant initial investment. **Maintenance and operation** costs for robotic mowers, however are relatively low: they are electric/battery-powered, so they do not require gasoline or oil changes. Users do need to replace cutting blades periodically (typically small razor-blades that cost perhaps $20 a set) and eventually replace the battery every few years. A CNET review pointed out that while you will not spend on fuel, you should budget for replacing blades and possibly the battery, which can be "fairly steep" costs on some models [18]. Some advanced models also required installation of

boundary wires (though newer ones use virtual GPS boundaries) and occasional troubleshooting if the robot gets stuck. Many robot mowers are designed to be weather-resistant and can operate in rain, though their cutting schedule might be interrupted by heavy rain (some have rain sensors to return to the dock). Aside from the mower itself, robotic weeding solutions are still in early stages - there are small solar-powered garden weeders (e.g. the *Tertill* robot for flowerbeds [28]), but for lawn weeds (like dandelions in the grass) no widely adopted robot exists yet. Weeding robots are more common in agriculture that in home lawns [29]. **Smart lawn sensors** - like soil moisture sensors connected to irrigation systems are another tech tool; these typically cost a few hundred dollars for a full smart controller setup and have minimal upkeep costs, often paying for themselves via water savings.

Maintenance Requirements: Robotic mowers are relatively low-maintenance compared to keeping a gas mower running. They automatically recharge themselves, many can run daily, clipping just a small length of grass each time (mulch mowing). Users must periodically clean the mower's underside, replace blades (usually a quick change done a few times a season), and ensure the perimeter system (if any) is intact. Batteries typically last 3-5 years before capacity drops, and a replacement battery might cost $100 - $300. Software updates are occasionally provided for higher-end models via apps. Overall, owners report spending much less time on lawn care - essentially just upkeep of the machine - once a robot mower is set up, compared to the hours previously spent mowing. This **time savings** is a huge selling point; as one survey noted, homeowners with robot mowers "got back valuable time - especially on weekends" [13]. For commercial use, multiple robotic mowers can be deployed to cover large areas, and maintenance might be handled by landscaping staff or the vendor. In terms of durability, robot mowers are built to handle weather and continuous use, but they do add a layer os teach that can occasionally malfunction (sensors, software glitches, etc., though these are improving with each generation).

Consumer Reception and Marketing: Robotic lawn care solutions have been marketed as the **future of lawn maintenance** - emphasizing convenience ("mow while you relax"), consistency (no missed mowing, the lawn always looks evenly cut), and quieter, emission-free operation. Ads often show people

enjoying their free time while the little mower quietly hums along. Environmental angles are also highlighted: battery-powered robots produce no direct emissions and use significantly less energy than gas mowers. Manufacturers like Husqvarna, IRobot (Terra), Worx and Robomow have educated consumers on how the technology works (many still find the concept novel. **Consumer reception** initially was cautious - early adopters in the 2010s were sometimes concerned about the performance (will it handle thick grass? hills? what about theft of the unit?). Over time, improvements such as better navigation (GPS-guided models that no longer need buried wires) [18] and anti-theft alarms have addressed many concerns. Reviews now show a mix of enthusiasm and realism: those with appropriate laws (fairly open, not too steep) and the patience to set up the system often **love the convenience**, while others with very complex yards or who enjoy mowing may see less value. Notable, many **mowing services themselves are eyeing robotic mowers** as a tool - some landscapers integrate robot mowers to handle open lawn areas while crew members focus on edging and detail work [15]. This hybrid approach is marketed to costumers as a way to get a perfectly maintained lawn more efficiently. Overall,. robotic lawn care is gaining acceptance: what was once seen as a high-tech gimmick is increasingly viewed as a practical solution, especially as price points gradually fall. As one industry report put it, "**homeowners are opting for autonomous solutions to upkeep their lawns**", leveraging smart tech for an easier experience [17]. With most major power-equipments brands now offering a robot mower model, it's likely these devices will become even more common in latter half of the 2020s.

In the context of lawn care, mobile autonomous robots extend these benefits to residential and commercial applications. They perform tasks such as mowing, edging, and weeding with precision, reducing resource consumption and environmental impact. These systems align with the principles of precision agriculture, bringing advanced solutions to smaller, more localized environments. The adaptability of mobile platforms to navigate uneven terrains and diverse conditions further underscores their versatility and practicality [30]. Figure 2.4 demonstrates a mobile autonomous robot navigating uneven lawn terrain, showcasing its adaptability and precision.

Figure 2.4: A mobile autonomous robot navigating uneven lawn terrain, highlighting its adaptability and precision [31]

The implementation of AI techniques, such as fuzzy logic, neural networks, and genetic algorithms, enhances the functionality of mobile robotic systems by enabling advanced navigation, task planning, and decision-making. These technologies ensure high productivity while requiring minimal human intervention. Despite their advantages, the scalability and integration of mobile autonomous systems into broader ecosystems remain challenges. Continued research and development in this area are essential to fully realize their potential in transforming agriculture and lawn care practices [30].

Mobile autonomous solutions represent a critical shift toward sustainable practices in agriculture and lawn care. By addressing labor shortages, improving productivity, and reducing environmental impact, these systems offer a pathway to more efficient and eco-friendly management of green spaces. Their ability to adapt to modern challenges ensures their relevance and viability as a cornerstone for future advancements in these domains.

Wheeled mobile robots have become integral to automating **residential and commercial estate maintenance** tasks such as lawn mowing, weeding, and edging. This chapter provides a comprehensive look at wheeled (and by comparison, tracked) robot locomotion for these applications, with emphasis on technical design evolution, industry prototypes, and the development of the Dandelion platform. Key locomotion challenges are examined alongside case studies of commercial products followed by a comparative analysis of wheeled

versus tracked locomotion. The design and manufacturing of the Dandebot's mobile base and attachments are then detailed, culminating in how Dandebot addresses unmet needs in yard maintenance.

2.4 Historical Development - Key Issues of Locomotion

Early robotic lawn care machines date back over half a century. The first known robotic mower, the **MowBot (1969)**, was a heavy autonomous lawnmower (57 kg) that pioneered features still seen today (boundary sensing, automated operation) [32]. However, such early attempts were **technologically limited** and not widely adopted [33]. Key locomotion issues included **power and traction** - battery and motor technology of the time limited running to 3 hours covering only 280 m^2 [32], and the weight raised concerns about turf compaction. Navigating uneven terrain and obstacles was rudimentary (bump sensors and random traversal), illustrating the challenge of robust locomotion control with the electronics available.

Through the 1990s and 2000s, progress in sensors, batteries, and micro controllers enabled a new wave of wheeled maintenance robots. **Husqvarna's Automower** (introduced in 1995) was the first solar-powered robotic mower [32], relying on a lightweight wheeled design and perimeter wires for guidance. These early models were predominantly **differential-drive wheeled robots** - two drive wheels and one or more free casters - which provided a simple, robust means of maneuvering on lawns. The **key locomotion challenges** remained obtaining sufficient traction on grass, handling slopes, and safely turning without getting stuck and or damaging turf. As an example, early generations often struggled near edges and could leave uncut strips along boundaries [34], highlighting limitations in both locomotion precision and deck placement.

By the 2010s, wheeled lawn robots became more common and sophisticated, featuring all-wheel drive for hills and better terrain handling. **Slope capability** improved markedly - for instance, the Husqvarna Automower 435X AWD can climb grades up to 70% (35° incline) [35]. This was achieved by advances in wheel traction design (treaded tires, optimized weight distribution) and multi-wheel drive. Meanwhile, some researchers and companies explored

tracked locomotion for outdoor robots to tackle rough terrain. Tracked prototypes offered superior traction on soft or uneven ground but introduced new problems like skid-steering resistance and mechanical complexity. Thus, historically, wheeled robots dominated residential yard care due to their simpler implementation and sufficient performance on maintained lawns, whereas tracks were considered for more extreme or unstructured terrains (e.g. heavy brush cutting on slopes).

In summary, the historical development of estate-maintenance robots has revolved around improving **locomotion reliability** - from ensuring wheels do not slip on damp grass to enabling robots to cover complex year layouts. Continuous refinement of wheeled designs (suspension, tire tread, drive power) has addressed many early issues, while tracking and positioning improvements (boundary wires to GPS-RTK) solved navigation rather than locomotion constraints [32].

2.5 Types of Wheeled Robots - Design Space and Industry Case Studies

Wheeled mobile robots for mowing and weeding come in various configurations tailored to different scales and tasks. The **design space of wheeled locomotion** spans from simple two-wheeled differentially driven platforms to four-wheel-drive systems with articulated steering Key categories include:

Differential Drive (Two-Wheeled) - Two powered wheels (left/right) and typically one or two caster wheels for balance. This is common in small lawn robots (e.g. Husqvarna Automower series, Friendly Robotics Robomow units) due to its minimal mechanical complexity. Differential drive allows in-place rotation (zero turning radius) by spinning wheels in opposite directions. For example, the Husqvarna Automower is a classic differential drive mower: its two rear wheels drive and steer the unit, while a front roller or caster balances it as shown in Figure 2.5.

Figure 2.5: Husqvarna Diff Drive Configuration [36]

This design is energy-efficient and sufficient for relatively even lawn terrain [37]. Many modern robotic mowers (e.g. Gardena Sileno, Bosch Indego) follow this paradigm, adding features like wheel trad for traction and wheel motors with encoders for precise speed control.

Four-Wheeled Skid-Steer - Some larger or all-terrain robots use four wheels in a skid-steer configuration (left-side wheels driven together, right-side together). This effectively behaves like a tracked vehicle in that steering is achieved by differential speed, causing the wheels to skid slightly during turns. Skid-steer wheeled designs provide better traction and load distribution than a 2-wheel drive does, which is useful for bigger commercial mowers. A case study is the BigMow by Belrobotics, a commercial robot mower that covers up to 20,000+ m^2; and it uses multiple drive wheels and floating cutting heads [38]. Skid-steer wheeled robots retain a low turning radius but can stress the grass during turns due to lateral wheel slippage.

Ackermann (Steered Wheels) - Less common in robotics (more common in ride-on mowers) uses a steering mechanism like a car. A robot with front-wheel steering and rear-wheel drive (or vice versa) can make smooth turns without skidding. For example, the *iRobot Terra* (a prototype robotic mower) had

a design resembling a small cart, with two driven wheels and a front caster, indicating it steered partly via differential drive and possibly gentle arcing turns [39]. Ackermann steering is generally used in larger outdoor robots or autonomous tractors, but for yard robots it's used sparingly because it increases the turning radius and mechanical complexity (steering linkages) without much benefit on open lawns where pivot turns are acceptable.

Omnidirectional (Holonomic) Wheels - Some research prototypes feature mecanum or omni-wheels that allow sideways movement. However, these are **rare in outdoor maintenance robots** because the complex wheels rollers or angled wheels have reduced traction on grass and can clog with debris. The design space includes holonomic drives for indoor service robots, but for mowing/weeding on uneven ground, holonomic wheeled platforms are not practical. Most industry examples favor differential or skid-steer drives for their robustness.

Hybrid Wheel-Track Systems - A few modern designs try to combine wheels and tracks. For instance, robot with retractable tracks or wheel-on-track systems exist in research [37]. In the maintenance domain, a notable example is the **Yarbo modular yard robot**, which uses rubber tracks over its wheels to gain traction on snow and slippery grass [40]. Yarbo can serve as a mower, snowblower, and leaf blower by swapping attachments, demonstrating an innovating blend of wheeled mobility (for speed) and track-like grip when needed. Such hybrid approaches remain limited to high-end units but illustrate the breadth of design possibilities.

Industry case studies highlight how these configurations are employed in real products:

Husqvarna Automower 435X AWD: A four-wheel-drive robotic mower with a pivoting chassis that handles complex lawns and slopes up to 70% [35]. It leverages AWD wheels for traction and stability on hills while maintaining a low ground impact on flat lawns. This product exemplifies pushing wheeled design to new performance extremes (all-wheel-drive, advanced sensors) for premium use cases.

Franklin Robotics "Tertill": A solar-powered weeding robot for gardens. Tertill is a small four-wheeled robot that roams garden beds, using its wheels

and a small string trimmer to cut weeds [28]. Its wheels are specifically designed to churn up the soil surface and disturb young weeds (in addition to the trimmer) [28]. Tertill is fully waterproof and lives in garden, illustrating how wheeled robots can be adapted for weed control in addition to mowing. Its design favors simplicity: four low-cost motorized wheels giving it mobility on loose soil, and it uses height sensors to distinguish weeds from taller plants.

In summary, wheeled robots dominate the market for lawn and garden maintenance due to their efficiency and adequate terrain handling for most properties. From tiny garden weeders to large-are lawn mowers, industry prototypes have explored various wheel counts and drive methods. Each design must balance **maneuverability, traction, and complexity** for its intended environment. Next, we compare the benefits and limitations of wheeled versus tracked locomotion in these contexts.

2.6 Benefits and Limitations of Wheeled vs. Tracked Locomotion

Wheeled and tracked locomotion each offer distinct advantages for outdoor robots, and understanding these is crucial when choosing a mobile base for tasks like mowing and weeding. Wheeled robots are known for their simplicity, speed, and energy efficiency on smooth terrain, while tracked robots excel in traction and stability on difficult terrain [37]. The following comparative analysis highlights key performance metrics:

Overall, wheeled locomotion is favored for most residential and light-commercial maintenance robots due to its efficiency, simplicity, and sufficient performance on typical lawns [37]. Tracked locomotion finds niche use in scenarios demanding extreme traction or stability (steep or uneven terrain) where wheels would falter [41]. Many modern designs seek a middle ground – for instance, all-wheel-drive robots or hybrids – to capture some benefits of tracks (traction) without their downsides.

Table 2.2: Wheeled vs Tracked Locomotion for Lawn Robots

Aspect	**Wheeled Locomotion**	**Tracked Locomotion**
Traction on Soft Ground	May slip or sink in mud	Excellent grip on soft terrain
Traction on Hard Ground	Strong grip on solid surfaces	Good grip but turning can drag
Turning Radius	Tight turns with little slip	Pivot turns need more force
Speed and Efficiency	High speed and runtime	Lower speed and battery life
Terrain Handling	Good on smooth terrain	Adapts well to uneven surfaces
Ground Pressure	Higher pressure on small area	Low pressure spread across track
Turf Impact	Minimal damage on grass	Can damage turf while pivoting
Mechanical Complexity	Simple with fewer parts	More parts and maintenance
Best Applications	Flat lawns and gardens	Slopes, rough or wet terrain

2.7 Building a Drivetrain

Robotic drivetrains commonly use brushless DC wheel motors for traction. For example, Clearpath's Husky platform uses four in-wheel brushless motors for a high-torque, all-terrain drivetrain [42]. Servo motors are generally used for steering or actuator joints rather than main drive (or continuous-rotation servos for small wheels). Control boards like ODrive can interface multiple motors; they often connect over CAN bus, enabling integration with ROS (e.g. ODrive's ROS2 CAN driver [43]). Essential sensors include an IMU and wheel encoders for odometry, GPS for absolute positioning, and LIDAR or depth cameras for mapping. For instance, Clearpath's Husky AMP comes with an integrated GNSS/IMU and 3D lidar for navigation [42].

Compute Modules: High-level processing is done on single-board computers (SoMs) such as Raspberry Pi, NVIDIA Jetson (Nano/Xavier/Orin), or similar. NVIDIA markets Jetson as a leading robotics AI platform [44]. Low-level control often runs on microcontrollers (e.g. ARM Cortex STM32, Arduino-class AVR or ESP32) that handle motor PWM, encoder reading, and fast safety loops. For example, hobby platforms use ESP32 with Arduino libraries for ROS integration and LiDAR sensing [45].

Sensors: Common choices are 9-DOF IMUs (measuring acceleration/rotation), quadrature wheel encoders, u-blox style GPS modules, and LIDAR units (e.g. Velodyne or Ouster). These feed into the onboard computer or microcontroller and are fused (e.g. combining GPS, IMU, lidar as in Clearpath's OutdoorNav [42].

Communications: Short-range buses like I2C and SPI connect sensors and low-speed peripherals. UART/USB link GPS or radios. For multi-device systems, CAN bus is popular for motor controllers and distributed electronics [43]. Ethernet or Wi-Fi is used for high-bandwidth devices (cameras) or cloud connectivity. Cellular or Wi-Fi radios connect the robot to remote apps or services (as used by Husqvarna mowers).

2.8 Image and Video Processing for Dandelion Detection

Detecting dandelions amid grass has been actively studied. Babiker (2020) developed a CNN-based vision algorithm to detect dandelion weeds in lawn grass and showed it accurately identifies dandelions under natural lighting conditions [46]. Follow-up work by Babiker, Bentahar and Xie (2024) proposed a heatmap-based CNN that locates the center of each dandelion. Their method achieved high precision and robustness to noise, outperforming classical segmentation and reducing labeling effort [47]. These studies highlight that deep learning can effectively distinguish dandelions from turf. For DandeBot, a YOLO (You Only Look Once) object detector is chosen for its simplicity and real-time performance. YOLO is a single-stage detector known for being fast and efficient in real-time vision tasks [48]. In practice, YOLO can be trained on a specialized weed image dataset (here, the WHEEL dataset of labeled weeds). Its ease of

use makes it suitable for deployment on an autonomous mower.

2.9 Low-Level and High-Level Control Architectures

Mobile robots typically separate low-level hardware control from high-level autonomy. Low-level control involves real-time motor commands, actuator drivers, sensor polling, and safety (e.g. emergency stop). High-level control handles navigation, path planning and decision-making. Many research platforms use ROS (Robot Operating System) as middleware. For example, Clearpath's Husky UGV ships with ROS2 pre-installed and provides drivers for its sensors and drives [42]. Husqvarna's research mower (Husqvarna Research Platform) also runs on ROS (compatible with ROS Kinetic/Indigo) [36]. In contrast, Boston Dynamics' Spot robot uses a proprietary API, but third-party ROS drivers exist (Clearpath released a ROS package for Spot in 2020 to let users leverage ROS tools) [49]. At high level, control software runs on the onboard computer (often Linux SBC). It may implement SLAM, state estimation, path planning, and user interfaces. Proprietary solutions (e.g. Boston Dynamics' internal control stack or Clearpath's OutdoorNav) are common. OutdoorNav, for instance, fuses GPS/IMU/LIDAR for localization and provides continuous path planning and obstacle avoidance [42]. Teleoperation modes allow switching to manual control when needed. Clearpath's OutdoorNav includes a teleoperation mode: an operator can remotely drive the robot via cellular link, viewing live sensor data and toggling autonomous mode on/off [42]. In summary, many systems use a two-tier architecture: microcontrollers or embedded controllers for real-time interfacing, and an SBC running ROS for autonomous behaviors.

2.10 Summary

This chapter reviews the key challenges and advancements in lawn maintenance, with a focus on the transition to robotic solutions. It highlights the environmental and economic issues of traditional methods, supported by a cost analysis from 2005 to 2025 that shows rising expenses and a shift from DIY to professional services. The increasing demand for automation and sustainability is backed by market trends and growing consumer awareness. Autonomous

lawn robots are explored for their efficiency, adoption, and ability to reduce labor while improving precision.

The chapter also compares wheeled and tracked locomotion using industry examples to guide DandeBot's design. It discusses core robotic components such as computing modules, sensors, and controllers, along with features like GPS navigation and remote access. The final section examines recent progress in AI-based weed detection, particularly the use of YOLO models for identifying dandelions, forming the foundation for DandeBot's autonomous weeding capability.

Chapter 3

Dandebot - Differential Drive Robot

This chapter covers the evolution of the Dandebot prototypes, focusing on the transition from early proof-of-concept models (v0.8 and v0.9) to the refined, production-ready design of Dandebot v1.0. It details the rationale behind the locomotion configuration, the mechanical design decisions, and the methods used to ensure design for manufacturing (DFM). Key elements such as chassis structure, modular tool integration, and the selection of materials and processes—like laser cutting, bending, welding, and surface finishing—are discussed. The chapter concludes with the machining processes, quality control procedures, and an overview of how various ready-made and custom components were integrated into the final assembly.

3.1 Earlier Prototype Versions - Dandebot v0.8 and v0.9

Before developing the current Dandebot v1.0, earlier prototypes v0.8 and v0.9 were built to experiment and validate the proof of concept. In addition to that the experiment with locomotion and a wide variety of conceputual mechanisms were tested. These iterative prototypes served as a testbed for refining the wheeled base and weeding/mowing tools:

Dandebot v0.8: The initial prototype featured a basic differential drive

configuration, equipped with two independent driving wheels and two additional castor wheels for support, as shown in Figure 3.1. It was constructed with a lightweight wooden frame and included all the essential components needed to test autonomous navigation capabilities and various low-level control algorithms. A detailed discussion of the control algorithms will be provided in a later chapter.

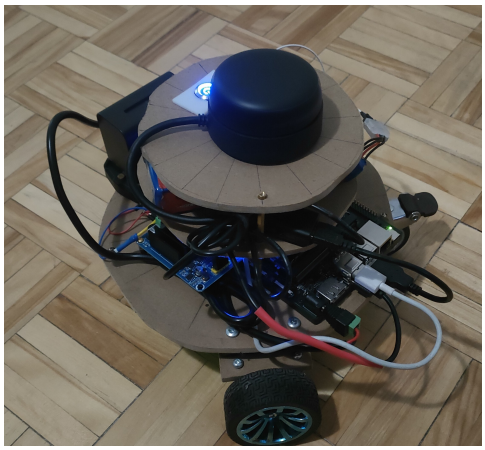

Figure 3.1: Dandebot v0.8

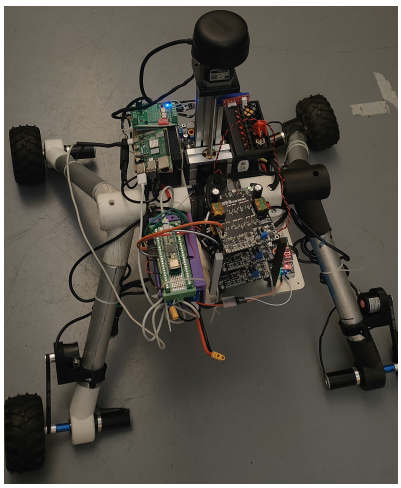

Figure 3.2: Dandebot v0.9

Dandebot v0.9: The subsequent version introduced enhancements in both traction and modularity. The setup as shown in Figure 3.2 was redesigned to incorporate a skid-steer mechanism, allowing for an evaluation of the differences in navigation and control performance. Additionally, the controllers were upgraded to ensure smoother operation and faster PID controller response for low-level control. The encoders were also improved, increasing the Pulse per Revolution for more precise locomotion.

These early versions were invaluable in identifying the key design criteria for Dandebot v1.0: sufficient traction without resorting to tracks, effective maneuverability for precise weeding, and a modular structure to handle different lawn-care tasks. Lessons learned (for example, v0.8's difficulty in maintaining straight paths on slight inclines, and v0.9's cumbersome tool-swapping process) directly influenced the final design described next.

3.2 Mobile base Development - Dandebot v1.0

With the insights gained from prototypes v0.8/0.9, the **Dandebot v1.0** was developed as a refined, production-ready mobile base capable of autonomous mowing and targeted weeding. This section details the design decisions and engineering process for the Dandebot v1.0 mobile base, focusing on locomotion configuration, mechanical design for manufacturing, and the fabrication methods used.

3.2.1 Configuration Selection - Holonomic vs. Differential Drive

Early in the design of Dandebot v1.0, various locomotion configurations were evaluated, included omni-wheel holonomic drives and conventional differential drives. A **differential drive** (non-holonomic) configuration was ultimately selected for Dandebot v1.0 as the optimal solution, providing a balance of maneuverability, simplicity, and outdoor performance. The choice can be justified by comparing the options:

Holonomic Drive: A holonomic drive allows movement in any direction without needing to rotate first, which is advantageous for indoor robots in tight spaces. However, for an outdoor lawn robot, this was deemed unnecessary - the tasks of mowing and weeding occur on open terrain where simply turning and driving forward is sufficient. Thus, the theoretical mobility benefit of holonomic motion did not translate well to the practical environment of lawns and gardens. Additionally, holonomic drives are mechanically more complex (requiring four driven wheels and more motors) and would consume more power, reducing battery life.

Differential Drive: A two-wheel or track differential drive proved to be highly maneuverable in its own right – it can pivot in place to change heading, effectively achieving zero-radius turns like a tracked vehicle [41]. This maneuverability is crucial for precise tasks such as edging along flower beds or repositioning to target a specific weed. Differential drive is also mechanically simpler (fewer drive motors than a holonomic system) and has well-understood control algorithms. Importantly, on the grass surface, the differential drive's performance is reliable: with proper tire tread, the wheels have enough friction

to pivot without excessive slippage or lawn damage. In Dandebot's testing, a differential drive could navigate tight spots (like around garden obstacles) by pivoting, negating the need for full holonomic motion.

3.3 Mechanical Design for Manufacturing

The mechanical design of Dandebot v1.0 focused on creating a sturdy, modular platform that could be manufactured with common metal fabrication techniques and easily maintained. The design is split into two major parts: (a) **the mobile base chassis** that houses the drivetrain and electronics, and (b) **the modular weeding and mowing mechanisms** that attach to the base. Emphasis was placed on design for manufacturing (DFM) – using techniques like sheet metal fabrication, standard profiles, and minimizing complex machining to allow efficient building of the robot.

3.3.1 Mobile base Chassis Design

The Dandebot's base is essentially the "body" of the robot. Key design considerations for the chassis included strength (to support attachments and endure outdoor use), weight distribution, ease of assembly and protection of internal components from the elements.

3.4 Design Procedure and Methodology

The design of Dandebot v1.0 followed a structured engineering process to ensure performance, manufacturability, and robustness for outdoor lawn maintenance. This section outlines the key procedures and calculations used in developing the final tracked differential drive platform.

3.4.1 Locomotion and Torque Requirement

To enable movement across uneven terrain and climb slopes up to 45 degree, torque calculations were performed to select suitable motors and gear reduction.

Robot mass: 65 kg
Slope angle: 45 degree
Effective drive pulley radius: 0.15 m

Gravitational acceleration: 9.81 m/s^2

Required torque per side:

$$\tau = \frac{M \cdot g \cdot r \cdot \sin(\theta)}{2} = \frac{65 \cdot 9.81 \cdot 0.15 \cdot \sin(45)}{2} \approx 33.7 \text{ Nm}$$

With a 1.5x safety factor:

$$\tau_{\text{design}} \approx 50.6 \text{ Nm per motor}$$

3.4.2 Speed and Power Estimation

Target speed: 1.0 m/s
Wheel radius: 0.15 m

Required angular velocity:

$$\omega = \frac{v}{r} = \frac{1.0}{0.15} \approx 6.67 \text{ rad/s}$$

Power per motor:

$$P = \tau \cdot \omega = 50.6 \cdot 6.67 \approx 337 \text{ W}$$

Total power: $\tilde{7}$00 W (including friction and margin)

3.4.3 Battery Runtime Estimation

Battery: 24V, 20Ah Li-ion
Energy: 480 Wh
Average load: 200 W

Estimated runtime:

$$t = \frac{E}{P} = \frac{480}{200} \approx 2.4 \text{ hours}$$

3.4.4 Track System Justification

A tracked system was chosen for:

- Better traction and weight distribution
- Stable slope handling (up to 45 degree)
- Precise zero-radius turning

3.4.5 Design for Manufacturing

Designs were optimized for DFM using:

- Laser-cut sheet metal chassis with minimal machining
- Modular bracket-based assembly (welded or bolted)
- Plug-and-play electrical and mechanical interfaces

3.4.6 Control Architecture

- **Low-level:** ODrive motor control with encoder feedback
- **High-level:** ROS 2 Navigation2 with DWB planner
- **Sensors:** LIDAR, IMU, camera for SLAM and detection

3.5 Technical Specifications

Table 3.1: Dandebot v1.0 Technical Specifications

Feature	**Specification**
Locomotion	Tracked differential drive
Weight	65 kg
Max Slope	45 degree
Motors	2x Brushless DC, 50 Nm each
Top Speed	1.0 m/s
Battery	24V 20Ah Li-ion
Runtime	$\tilde{2}$.4 hours
Sensors	LIDAR, IMU, Camera
Compute Units	Jetson, Raspberry Pi, ESP32
Software Stack	ROS 2 Navigation2, DWB planner
Modules	Swappable mowing & weeding tools
Fabrication	Laser cutting, welding, milling
Materials	Mild steel & 6061-T6 aluminum

Chassis Structure: The chassis is built primarily from **sheet metal and aluminum extrusion** elements. We selected **6061-T6 aluminium alloy** for many structural parts due to its balance of strength, light weight, and availability in plate and angle forms. The base frame consists of a bottom plate and reinforcing ribs cut from mild steel sheet (using waterjet or laser cutting for precision), which are then bent or welded into shape. Using sheet metal allowed us to integrate features like motor mounts, slots for wheel axle adjustments, and enclosure tabs directly into the cut design, reducing the need for separate brackets. The overall form is a low rectangular hull, with cutouts for wheels and tool modules. The design was optimized with finite element analysis (FEA) to remove excess material in low-stress areas, keeping the frame lightweight while maintaining rigidity under load.

Drive Wheel Mounts: Each drive shaft for the tracked system is supported by a robust steel axle mounted securely to the chassis through precision-machined flange bearings bolted directly onto reinforced sidewalls. To achieve a compact and efficient power transmission, we employed a bevel gear system

positioned perpendicular (90 degrees) to the track shafts. This right-angle drive configuration significantly reduces space requirements inside the chassis, improving accessibility for maintenance and enhancing the overall compactness of the robot.

The powertrain consists of two high-performance brushless DC motors as shown in Figure 3.3 coupled to integrated gearboxes that reduce speed while amplifying torque. These motors are mounted perpendicular to the track shafts, transmitting power via precision-cut bevel gears directly to the track drive sprockets. The gear assembly ensures reliable torque delivery and minimal backlash, critical for responsive and precise control of the tracked locomotion system.

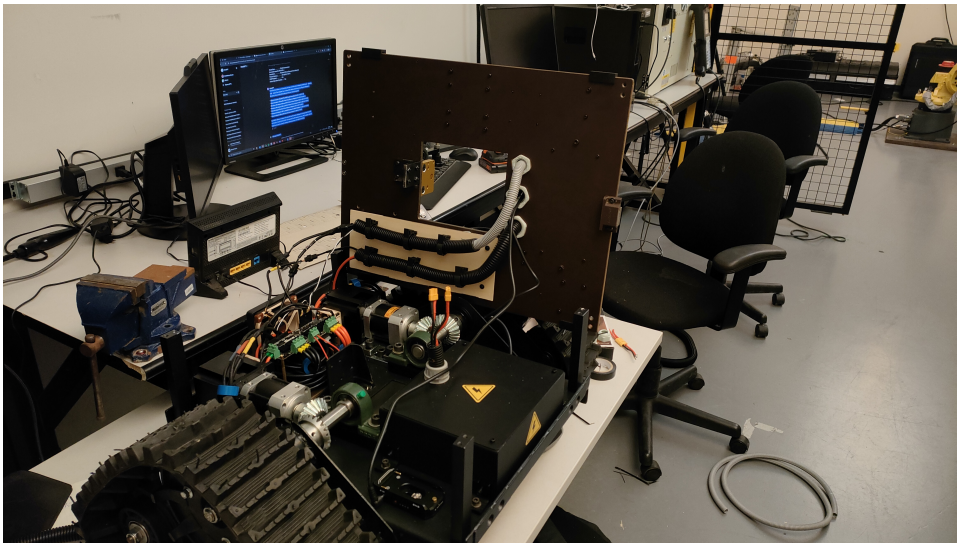

Figure 3.3: Dandebot v1.0 Drive Configuration

For ease of maintenance and assembly, the design features removable top access panels on the chassis, allowing convenient inspection, servicing, or replacement of motors, gearboxes, and bearings without extensive disassembly. Motor mounts are fabricated from sturdy aluminum brackets. While these mounts can be welded directly to the chassis using precise AC TIG welding, bolted connections with reinforced nut inserts are recommended for simpler manufacturing, easier replacement, and reduced risk of weld-induced warping.

The tracks themselves interface with sprockets rigidly fixed to the main drive

shafts. This setup ensures consistent engagement with the track links, minimizing slip and wear even under demanding operating conditions. Bearings and bevel gear housings are carefully sealed against dust and moisture, enhancing system durability for prolonged outdoor use.

Modularity Interfaces: The base includes mounting interfaces (hardpoints) for attaching the modular mechanisms. For instance, at the center front of the chassis there are quick-attach brackets where the weeding tool or mowing deck can be bolted on. Power and data connectors are standardized so that either module can plug into the base's electrical system. This modular approach influenced the chassis design – we reinforced those areas of the frame where heavy modules connect, and positioned the center of gravity low and central so that swapping modules (which changes weight distribution) won't significantly affect stability. The chassis also has tie-down points and lifting handles, which are helpful both in manufacturing (fixturing during welding/powder coating) and in practical use (carrying the robot or strapping it during transport).

In summary, the mobile base was designed for robustness and manufacturability: a mild steel-sheet chassis that is weldable and powder-coatable, using standard bearings and hardware, and providing a stable platform for the robot's operations. This design can be efficiently fabricated with common processes as described later.

3.5.1 Modular Mechanism - Weeding and Mowing Attachments

One of Dandebot's distinguishing features is its ability to perform multiple maintenance tasks. This is achieved through two main attachments: a mowing module and a weeding module, which can be attached to the base as needed. Both modules were designed to be as modular and DFM-friendly as the base, using common components where possible.

Mowing Module: The mowing mechanism is designed as a compact and modular attachment that mounts directly to the underside of the Dandebot base when needed. It consists of a rotating blade or a spinning head equipped with string trimmer lines for enhanced safety, contained within a protective steel deck housing. The mowing deck itself is precision laser-cut from either

stainless steel or powder-coated steel sheet to effectively resist corrosion from moisture and grass clippings.

The power to the mowing blade is delivered via a dedicated electric motor (distinct from the tracked drive motors), which is attached securely to the deck housing through a standardized flange mount. For ease of manufacturing and replacement, the blade used is a commercially available mulching blade, custom-cut to the appropriate length to ensure compatibility and efficient cutting performance.

The cutting height of the mowing mechanism is easily adjustable by adding or removing spacers at the attachment points. This allows precise vertical positioning of the blade relative to the ground, catering to various lawn conditions and user preferences. The mowing module securely connects to the Dandebot base using four quick-release pins, ensuring rapid attachment and removal without specialized tools. The connection is made via a robust, waterproof plug supplying the module with both power and control signals.

When the mowing module is detached, the robot operates with reduced weight and power consumption, enhancing maneuverability and efficiency during specialized tasks such as weeding. This modular design philosophy aligns with multifunctional yard care concepts, inspired by industry-leading platforms like Kobi and Yarbo. The emphasis has been placed on secure yet convenient modularity, enabling rapid interchangeability between mowing, weeding, and other lawn-care tasks within minutes.

Weeding Module: The weeding mechanism is more complex, as it targets individual weeds (like dandelions) in the lawn. The Dandebot's weeding module is a lightweight robotic arm or plucking device that can identify and remove weeds either by pulling them out or applying a concentrated treatment. Mechanically, this module includes a small 2-DoF arm as showin in Figure X with an end-effector designed for weed removal. For example, one approach is a forked prong that stabs into the soil and lifts the weed root out. Another approach trialed was a rotating brush or auger to uproot the weed. In either case, the module attaches to the front of the base, extending forward so it can reach the ground just in front of the robot. The module's structure is built from CNC-machined aluminum and steel plates and source some readymades like pulleys and belts. We decided to allow some 3D printed plastic components

here (using durable ABS or PETG) for intricate parts of the weeding tool that are not load-bearing initially to test the proof of concept and proceeded to produce stainless steel parts, due to its non-corrosive nature.

Designing these mechanisms with manufacturing in mind involved keeping parts geometrically simple for fabrication. For instance, flat plate components for the weeding arm linkages were waterjet-cut aluminum, avoiding complex 3D contours. Housings that required 3D shape (like a cover around the weed plucking mechanism) were allocated to 3D printing rather than expensive machining. Wherever possible, we used COTS (commercial off-the-shelf) parts: e.g., a standard 12-inch mower blade, standard bearings, bolts, and even sections of pre-made robot arms or linear actuators to reduce custom fabrication. This not only eases manufacturing but also future maintenance or replacement by the end user.

By partitioning Dandebot's functionality into the base and modules, we achieved both functional versatility and manufacturing practicality. Each module can be worked on independently in production – for example, the mowing decks can be stamped and assembled in one area while bases are welded in another, then they come together in final assembly.

3.6 Machining and Fabrication Processes for Dandebot v1.0

3.6.1 Machining

Machining operations involve precision manufacturing of structural and functional components, including milling and turning processes:

Milling:

Motor Brackets: Precision milling ensures accurate mounting dimensions and alignment for motor attachment as shown in Figure 3.4.

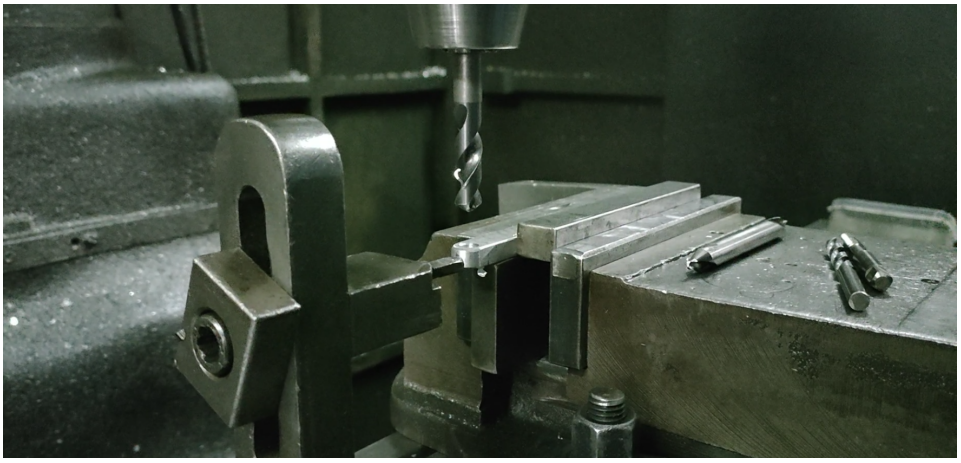

Figure 3.4: Example Process for Milling

Fixture Components: Certain couplers and structural parts undergo milling to meet critical dimensional tolerances and surface finishes.

Machine Base: Precisely milled to include holes, taps, and slots from 3 mm thick mild steel sheets for secure component fixing, ensuring structural rigidity and durability.

Track Fixtures: Precision-machined to accommodate belts, drive wheels, and two idler pulleys, ensuring proper track alignment.

Digging Mechanism Components: Couplers, ejector fixtures crafted primarily from mild steel, aluminum, and stainless steel.

Cutting Blades: Machined from stainless steel or hardened mild steel for both digging and mowing applications.

Turning:

Ejectors: Cylindrical ejectors for the weeding module are produced through precise turning processes, ensuring consistent dimensions and smooth operation. As shown in Figure 3.5, a conventional lathe is used for all the machining processes.

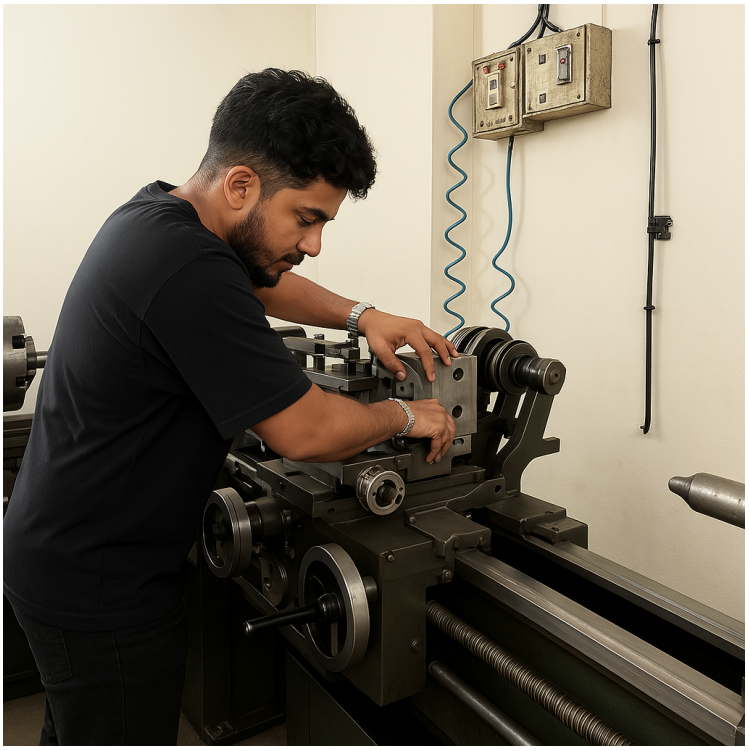

Figure 3.5: Example Process for Turning

Couplers: Some couplers in the digging mechanism are turned to guarantee proper fit and functionality.

3.6.2 Fabrication

Fabrication processes include laser cutting, bending, and welding operations for the creation and assembly of metal structures and enclosures:

Laser Cutting:

Enclosure Sheets: Laser-cut precisely from mild steel sheets (1.6 mm) for battery enclosures, machine base, and other structural components. All the laser cutting processes were done in the machine as shown in Figure 3.6.

Figure 3.6: Laser Cutting Machine

Base Plates: Laser-cut from thicker mild steel sheets (3 mm) to serve as strong foundational elements.

Fixture Components: Accurate cutting ensures precise fit and alignment of track fixtures and brackets.

Bending:

Sheet Metal Components: Post laser-cutting, sheets undergo precise bending operations to form enclosures, brackets, and structural supports. An example bending process of an enclosure sheet can be seen in Figure 3.7.

Figure 3.7: Example Process for Fabrication

Chassis and Enclosure Flanges: Bending operations create rigid flanges, providing secure mounting points and improving structural strength.

Welding:

Assembly Welding: Structural welding performed post-assembly to permanently join metal components, reinforcing the chassis, enclosures, and fixture components. Every welding joint was done by either a TIG or a MIG welding machine and the process can be seen in Figure 3.8.

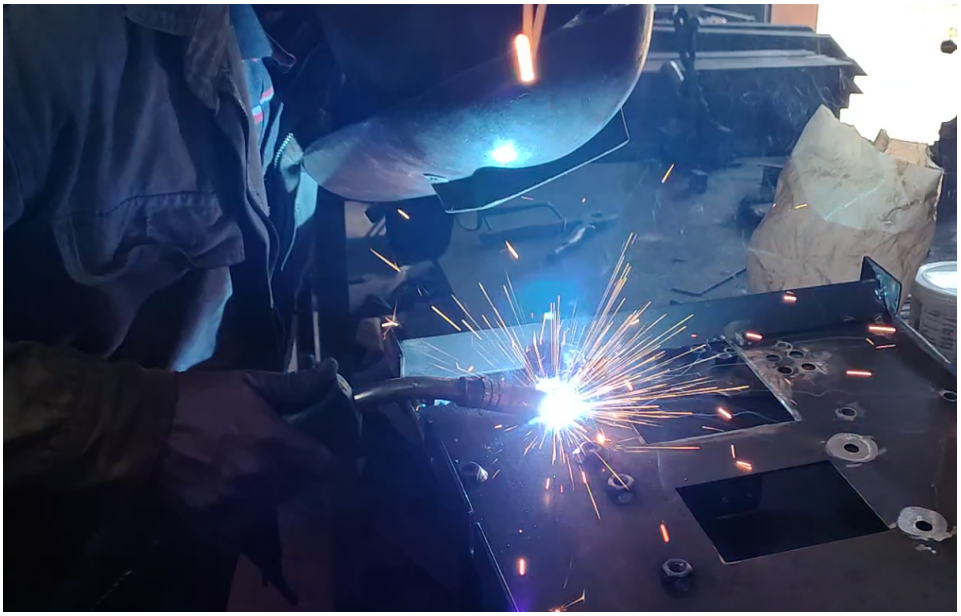

Figure 3.8: Example Process for Welding

Chassis and Fixture Integrity: Ensuring that all joined components withstand operational loads and environmental exposure through meticulous welding.

3.6.3 Heat Treatment

Blades: Digging and mowing blades undergo heat treatment processes (hardening and tempering) to significantly enhance durability, wear resistance, and cutting performance.

3.6.4 Surface Finishing - Powder Coating and Anodizing

Powder Coating: All mild steel sheets used for enclosures, base plates, and fixtures receive a durable polyester powder coat for corrosion resistance, environmental protection, and aesthetic appeal.

Anodizing: Aluminum components including motor brackets, certain fixtures, and precision machined parts receive anodizing treatment, enhancing surface hardness, corrosion resistance, and providing a professional finish.

3.6.5 Final Assembly and Quality Control

Assembly Integration: All components, including pre-machined, fabricated, powder-coated, and anodized parts, are assembled meticulously using standardized fasteners. Some of the assembly pictures can be seen in Figure 3.9 and 3.10.

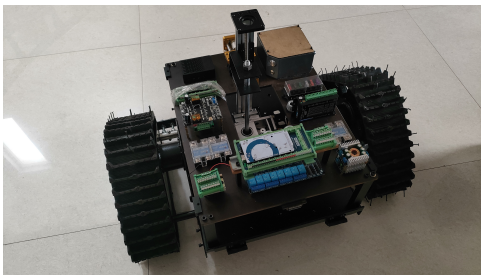

Figure 3.9: Assembly without side enclosure

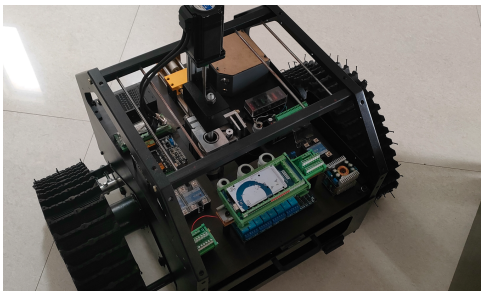

Figure 3.10: Assembly with side enclosure

Quality Control Checks: Each sub-assembly and the final assembled robot undergo rigorous quality checks, including functional tests of motors, drive systems, modules, and structural assessments to ensure overall product reliability.

Ready made Components: Fasteners, belts, pulleys, bevel gears, and bearings are sourced as standard ready made items to streamline the manufacturing and assembly process, ensuring consistency and reliability in final assembly.

3.7 Summary

This chapter describes the development of Dandebot, a modular differential drive robot built for autonomous lawn care. Early prototypes (v0.8 and v0.9) were used to test locomotion, traction, and modularity, leading to the design of the production-ready Dandebot v1.0. A differential drive system was chosen for its mechanical simplicity, outdoor reliability, and ability to execute zero radius turns, making it suitable for navigating complex garden layouts.

The robot's design emphasizes manufacturability using standard materials and processes such as laser cutting, milling, welding, and turning. Its sheet metal chassis houses brushless motors, bevel gears, and modular interfaces for mowing and weeding tools. The mowing deck is electrically adjustable, while the weeder

includes a robotic end effector for precise weed extraction. The chapter also covers fabrication steps, surface finishes, and use of standard components to ensure durability, maintainability, and scalable production.

Chapter 4

SoM with Sensors and Actuators

This chapter provides a clear overview of the key hardware components in a fully electric, AI-driven lawn care robot. We focus on sensors, actuators, the system-on-module, microcontroller-based controllers, communication methods, and the role of ROS (Robot Operating System). Each component is described in simple terms, explaining how it works and its role in the robot's operation. These components serve as the "eyes", "muscles", and "brain" of the robot, enabling autonomous mowing, weeding, and edging.

4.1 Sensors: The Robot's "Eyes" and Feedback Devices

Sensors allow the robot to perceive its environment and its own motion. They act as the robot's eyes and ears, collecting data that the robot's control system uses to make decisions. In this robot, important sensors include a LIDAR, a stereo camera, limit switches, incremental encoders, and a voltage sensor. Each sensor has a specific way of operating and a dedicated role:

4.1.1 LIDAR (Light Detection and Ranging)

How it works: LIDAR is an active optical sensor that measures distance by timing laser pulses. It emits rapid pulses of laser light and waits for them to bounce off objects and return. By measuring the time of flight of each pulse

(the time taken for the light to go out and back), the sensor computes the distance to that object.

Role in the robot: The LIDAR acts as one of the robot's primary eyes for mapping and obstacle detection. It gives the robot a sense of how far nearby objects are. For example, as the lawn robot moves, LIDAR can detect obstacles (like a tree, a garden hose, or a person) by returning distance measurements in those directions. The robot's control system can use this data to avoid collisions and to understand the layout of the yard.

4.1.2 Stereo Camera

How it works: A stereo camera is a pair of cameras separated by a small distance (baseline) that mimic human binocular vision. Both cameras take images at the same time, and by comparing these two images, the system can perceive depth. This works through the principle of triangulation: for any feature in the scene, its projection appears at slightly different positions in the left and right images. The difference in position (called disparity) is larger for closer objects and smaller for farther objects. Given the geometry (the cameras' separation and focal parameters), the stereo vision algorithm computes the distance to points by finding corresponding features in the two images [50]. In simple terms, the stereo camera provides a depth map by comparing two images – much like our brain judges depth by comparing what our left and right eyes see.

Role in the robot: The stereo camera is another "eye" of the robot, providing rich visual information including both color and depth. While LIDAR gives precise distance measurements, the stereo camera provides actual camera imagery and can detect things LIDAR might not (like the color/texture of plants). The AI on the robot can use stereo camera feeds to recognize objects – for example, distinguishing weeds from grass or detecting the edge of a flowerbed. Moreover, because it provides depth perception, the robot can estimate how far visual features are. Stereo vision thus complements LIDAR by offering vision-based obstacle detection and scene understanding. It is especially useful in outdoor environments where lighting varies, since modern stereo cameras are designed to handle a wide range of lighting (and unlike some laser sensors, stereo vision is not affected by sunlight glare [51]. Overall, the stereo camera

enables the robot to see its work area in a human-like way, which is invaluable for tasks like identifying weeds for removal.

4.1.3 Limit Switches (for 2-DOF Weeder Homing)

How it works: A limit switch is a simple electromechanical sensor that triggers when a moving part reaches a certain position. It typically consists of a small lever or button actuator and an internal electrical switch. When the robot's mechanism physically presses the lever, the switch closes (or opens), sending a signal. In industrial usage, "limit switches are used to detect the presence or absence of an object" and to define the endpoint of travel [52]. They are essentially binary sensors (triggered or not triggered) activated by contact. For example, in our 2-DOF weeding arm, a limit switch might be placed at the top of the arm's range. When the arm moves upward and hits that switch, it knows it has reached the home (reference) position.

Role in the robot: The limit switches on the two-degree-of-freedom (2-DOF) weeder serve as homing sensors. Before the robot starts weeding, it needs a reference point for the weeder's moving parts (such as the vertical up-down motion and maybe a horizontal extension). The robot will drive the weeder mechanism slowly until a limit switch is hit, which signals that the mechanism is at a known "zero" position. This homing process ensures that the stepper motor controlling the weeder starts from a calibrated position each time. Beyond homing, limit switches also act as safety stops – preventing the mechanism from moving beyond its physical limits. They are simple and reliable; as an electromechanical device operated by physical force, a limit switch provides a clear on/off indication when the weeder has moved to its end-stop [52]. In summary, the limit switches protect the weeder assembly and provide positional references, allowing the robot to consistently and safely control the weeding tool's motion.

4.1.4 Incremental Encoders (Wheel/Motor Encoders)

How they work: An incremental encoder is a sensor attached to a rotating shaft (for example, a wheel axle or a motor shaft) that generates pulses as the shaft turns. Typically, it consists of a disk with alternating transparent and opaque segments and an optical sensor, or a magnetic wheel with sensor,

producing electrical pulses. Two output channels (A and B) are offset in phase (a quadrature signal) so that by watching the order of pulses the system can determine the direction of rotation. Every time the wheel rotates a tiny fraction, the encoder emits a pulse. By counting these pulses, the robot's controller can measure how much the wheel has turned [53]. Because incremental encoders only report changes (relative movement) and not absolute position, the system usually counts from a known start point (which could be set via a limit switch or simply zero at startup). Encoders report movement almost instantaneously, allowing precise tracking of speed and position changes in real time.

Role in the robot: The incremental encoders on the robot's drive wheels (or drive motors) give the robot a sense of its own motion – analogous to an odometer and speedometer in a car. By counting pulses, the robot knows how far it has traveled and how fast each wheel is spinning. This feedback is crucial for wheel odometry (estimating the robot's position on the lawn) and for speed control. For example, if the robot commands its wheels to move a certain distance, it will use the encoder counts to determine when that distance is achieved. If one wheel begins to slip or drag, the encoder rate will drop and the controller can compensate by driving that wheel motor harder or adjusting the other wheel to maintain straight travel [53]. In summary, encoders close the loop for the drive system, providing the data needed for accurate motion control and coordination between motors. Without encoders, the robot would have to run its wheels open-loop and would quickly lose track of how far it moved. With them, it achieves precise movement and can execute complex paths reliably because encoders allow precise measurement and control of position and velocity [53].

4.1.5 Voltage Sensor (for Battery Management System)

How it works: The voltage sensor is an electronic measuring device that monitors the battery's electrical potential (voltage). In a Battery Management System (BMS), a voltage sensor typically is a simple circuit (often a voltage divider feeding an analog-to-digital converter) that continuously measures the battery pack's voltage. If the robot has a multi-cell battery, the BMS might measure each cell's voltage. The readings are used to estimate the state of charge of the battery – much like checking the pressure in a tank to guess how full it is. Battery monitors often act like a "fuel gauge" for the battery, using

voltage to infer charge level[54]. In addition, the BMS uses the voltage sensor to detect over-voltage (when charging) or under-voltage (when the battery is almost empty) conditions.

Role in the robot: The voltage sensor keeps the robot informed about its battery status. Since this lawn robot is fully electric, maintaining the battery within safe limits is critical. The sensor feeds the BMS data about real-time voltage, which the BMS uses to prevent the battery from over-discharging or over-charging [54]. For instance, if the battery voltage drops near a threshold, the robot might decide to return to its charging station or shut down non-critical functions. In essence, this sensor allows the robot to manage its energy: it can estimate remaining runtime (similar to a laptop showing battery percentage) [54] and ensure longevity of the battery by avoiding deep discharge. Think of it as the robot checking its fuel tank – the voltage sensor's input helps the system behave intelligently regarding when to recharge. Moreover, because the BMS can also use voltage (along with current sensors, if present) to detect battery health issues, it contributes to the overall reliability and safety of the robot's power system. In summary, the voltage sensor in the BMS is a small but crucial sensor that helps the robot autonomously monitor its own power supply, ensuring it doesn't run out of power in the middle of a job and that the battery stays within safe operating conditions.

4.2 Actuators: The Robot's Motors and Moving Parts

Actuators are the components that perform physical actions and they are the "muscles" of the robot, translating electrical signals into movement. In our lawn robot, the main actuators are the drive motors (for locomotion), the motorized 2-DOF weeder mechanism, and the mower motors. Specifically, the design uses servo motors for wheel locomotion, a stepper motor for moving the weeding tool up and down (Z-axis), and a DC brushed motor for rotating the weeding tool (along the vertical axis). Each type of motor is chosen for a reason, and each works a bit differently:

4.2.1 Servo Motors (for Locomotion Drive)

How they work: In a broad sense, a servo motor is a motor equipped with feedback and a controller, enabling precise control of position, speed, or torque. Unlike a regular DC motor that spins freely when powered, a servo motor system constantly monitors its own position (using an encoder or potentiometer) and adjusts its drive input to reach the target position or speed set by the controller. Many servos (for example, in hobby robotics) have an internal control circuit: you send a command (like a PWM signal) specifying a desired angle, and the servo's internal electronics drive the motor until a built-in sensor (like a potentiometer) indicates the angle is reached. In the case of our lawn robot's wheels, the "servo motors" are likely brushless DC motors with encoders controlled by an external servo driver (the ODrive board, discussed later) [55]. They operate in closed-loop mode: if the robot needs the wheel to spin at 50 RPM, the controller reads the wheel's encoder and increases or decreases motor power to lock on that speed. If an uphill slope slows the wheel, the servo controller will sense the speed error and apply more power to maintain the commanded speed. In short, servo motors combine a motor + feedback + control logic to give high-precision, controlled motion.

Role in the robot: The servo-driven locomotion motors propel the robot and steer it with accuracy. By using servo control on the drive wheels, the robot can maintain exact speeds on each wheel and can execute motions like straight lines or turns with reliable precision. This precision is necessary for systematic lawn mowing (covering the area in neat, non-overlapping paths) and for delicate maneuvers around obstacles or along edges. Another advantage is torque and speed control: servo-controlled motors can provide the necessary torque when the robot encounters thick grass or inclines, without losing track of position. For example, if one wheel starts to slip on wet grass, the encoder feedback will show unexpected acceleration; the controller can reduce power to that wheel to regain traction. In essence, servo motors give the robot "smart wheels", the robot doesn't just power its wheels forward blindly, it actively controls their motion to follow the planned trajectory. This makes the locomotion robust and responsive. Additionally, servo motors can smoothly ramp speeds up and down, which is gentle on the hardware and saves energy. The result is a self-driving lawnmower that moves efficiently and accurately, thanks to servo-driven locomotion (the combination of powerful motors and continuous

feedback control).

4.2.2 Stepper Motor (for 2-DOF Weeder Z-axis)

How it works: A stepper motor is a special type of brushless DC motor that moves in discrete steps rather than continuous rotation. Internally, it has multiple electromagnetic coils arranged in phases around a rotor with teeth or permanent magnets. By energizing the coils in sequence, the rotor is pulled from one stable position to the next. Each pulse of current advances the motor by one fixed angle step (for example, 1.8° per step, which would be 200 steps for a full rotation) Because steps are repeatable and known, you can rotate the motor by an exact amount by sending a specific number of pulses [56]. Notably, a stepper motor holds its position firmly when stopped at a step (provided the coils remain energized), resisting external torque up to a limit. Standard stepper motors do not require an encoder for position in many applications because if you count the steps you've commanded, you know the position – as long as the motor hasn't skipped any steps due to overload. They operate open-loop; however, at the start we often use a homing sensor (like a limit switch) to define a zero reference. After homing, the system assumes each step command moves the motor reliably by the step angle [56]. Stepper motors excel at low-speed, precise positioning tasks, but they do consume current to hold position and can lose synchronization if overloaded.

4.2.3 DC Brushed Motor (for Digger Rotation along Z-axis)

How it works: A brushed DC motor is a traditional electric motor design that runs on direct current and uses internal brushes and a commutator to switch current in the armature as it rotates. When you apply a DC voltage, current flows through coils on the rotor; these coils become electromagnets that interact with fixed magnets in the stator, producing torque and rotation. The commutator is a segmented copper ring attached to the rotor, and stationary carbon brushes press against it. As the rotor spins, the commutator automatically flips the direction of current through each coil at the appropriate time, sustaining continuous rotation. In essence, the mechanical commutation ensures the magnetic poles keep chasing each other around, pulling the rotor

forward. The speed of a brushed motor is controlled by the applied voltage (higher voltage - faster rotation), and the torque is roughly proportional to current [57]. One reason these motors are popular is simplicity: "Brushed DC motors operate on direct current, and their speed can be easily controlled by altering the applied voltage". However, the brushes do wear over time and create some electrical noise due to sparking. Despite that, they are inexpensive and provide a good power-to-weight ratio, especially for continuous rotation tasks.

Role in the robot: The DC brushed motor in the weeder is responsible for the rotation of the digging tool around its vertical axis (the Z-axis). Think of this as the motor that spins the weed extractor – for example, turning an auger or a small blade that cuts into the soil around the weed. This is a continuous rotation movement, well-suited to a brushed DC motor's characteristics. When the robot positions the weeder above a weed and lowers it using the stepper, it then uses the brushed motor to spin the digger. The advantages of a brushed DC motor here are its simplicity and high speed: it can spin rapidly to sever roots or drill into the ground, and controlling its speed is as easy as adjusting the voltage or using a PWM signal from a motor driver. Brushed motors also have good starting torque, so they can begin turning the weeder even if there's some resistance (soil can offer significant friction). Because the task (weed extraction) doesn't demand precise angle control – it just needs to rotate continuously to break up soil – a brushed motor is a sensible, cost-effective choice. After the weed is uprooted, the robot can simply stop powering the motor to halt the spinner. Maintenance on this motor would mostly involve checking brushes eventually, but for a relatively low-cost part, that is acceptable in exchange for effective weed removal action. In summary, the brushed DC motor gives the robot a simple "spinner" tool: it reliably delivers rotational power for digging and edging functions, complementing the stepper (which handles up/down positioning) by adding the necessary twisting motion to uproot weeds or trim edges.

4.3 System on Module (SoM): The Robot's Computing "Brain"

What is a SoM? A System-on-Module is essentially a compact computer board that includes a processor, memory, and essential interfaces, designed to plug into a larger carrier board. It's like the "brain" of a computer, condensed onto a small card. A SoM typically contains the CPU (and GPU if available), RAM, storage (or storage interfaces), and often common connectivity (like USB, Ethernet controllers) on one module. The idea is to integrate all core computing components so that developers can place this module into their device rather than designing a computer from scratch. The carrier board then provides additional connectors, power regulation, and any custom peripherals the specific application needs. In our robot, the chosen SoM is the NVIDIA Jetson Orin Nano, which is a powerful module aimed at AI and robotics applications.

4.3.1 NVIDIA Jetson Orin Nano

What it is and how it works: The Jetson Orin Nano is a state-of-the-art SoM produced by NVIDIA for edge AI computing. It contains an NVIDIA Ampere-architecture GPU with built-in tensor cores (special units for accelerating deep learning computations), along with an 6-core ARM Cortex-A78AE 64-bit CPU, and it has LPDDR5 memory all on the module. Despite its small size (fit-in-your-palm), it delivers up to approximately 67 trillion operations per second (TOPS) of AI performance, meaning it can perform huge numbers of neural network calculations per second while consuming relatively low power (in the range of 7W to 15W typical, up to 25W max) [58]. The Jetson runs a full Linux operating system (Ubuntu-based) and the ROS middleware, and it is equipped to handle multiple sensors – for example, it supports up to 4 camera inputs, USB devices, and more. Essentially, it's a supercomputer for AI tasks in a credit-card-sized package. The Orin Nano specifically is optimized for running modern AI models for computer vision, navigation, etc., on the robot, providing efficient on-board computation without needing a connection to cloud or external computers.

Why it's used in the robot: The Jetson Orin Nano serves as the central processing unit of the autonomous lawn robot. All the sensor data (from LIDAR, cameras, encoders, etc.) is funneled to the Jetson, which then runs advanced algorithms (like object detection, path planning, and decision-making AI). For example, the stereo camera feed will be processed by neural networks on the Jetson to identify weeds or to detect the boundaries of the lawn. The Jetson's GPU and AI accelerators are critical for this – they allow real-time image recognition and even potentially segmentation of the terrain (distinguishing grass vs flower bed). Additionally, the Jetson coordinates the high-level navigation: it can build a map from LIDAR data and plan an efficient mowing path. This requires significant computation, which the Orin Nano can handle due to its high throughput and parallel processing capability. It's effectively the robot's brain, enabling it to think and make decisions. Another reason to use a module like Jetson is energy efficiency: it provides a lot of compute per watt, important for a battery-powered device. Also, development is accelerated by this SoM because NVIDIA provides a rich software stack (libraries and SDKs for AI and robotics) that the thesis can leverage – for instance, the Jetson supports the ROS environment out-of-the-box and has GPU-accelerated libraries for image processing. In summary, the Jetson Orin Nano SoM is chosen to give the robot the necessary computational muscle for AI-powered autonomy while fitting within the size, weight, and power constraints of a lawn mower robot. It's a self-contained "computer core" that makes the robot intelligent, handling everything from running the operating system to crunching sensor data into actionable plans.

4.4 Controllers: Microcontrollers and Motor Drivers for Real-Time Control

While the Jetson Orin Nano is the high-level computer, the robot also relies on dedicated controller boards to interface with sensors and actuators at a low level, especially for real-time tasks that require precise timing. These controllers include microcontroller units (MCUs) and motor control boards that act as the robot's "nervous system," ensuring that commands from the Jetson are executed by the motors and that sensor readings are collected reliably.

The controllers in this robot are:

- **Teensy 4.1** (ARM Cortex-M7 microcontroller board)
- **ODrive v3.6** (dual-channel motor driver for brushless motors)
- **Arduino Mega 2560** (ATmega2560 microcontroller board)

Each plays a different role in the control architecture.

4.4.1 Teensy 4.1 (ARM Cortex-M7 MCU)

What it is: The Teensy 4.1 is a high-performance microcontroller development board. It features an NXP i.MX RT1062 processor which has a 600 MHz ARM Cortex-M7 core (32-bit) with floating point and DSP extensions. This means it runs extremely fast for a microcontroller – the Cortex-M7 can execute multiple instructions per cycle and has generous memory, making it capable of complex, real-time computations on the microsecond scale [59]. The Teensy 4.1 also provides a lot of I/O: digital and analog pins, several serial communication ports (UART, I2C, SPI), and even an SD card slot and Ethernet interface available. Importantly, even though it's powerful, it's still a microcontroller (no operating system by default), so it starts up instantly and is highly predictable in timing (good for real-time control loops).

Role in the robot: The Teensy 4.1 likely serves as the real-time control hub for things that the Jetson (running Linux) cannot reliably do with precise timing. For instance, tasks such as reading quadrature encoder signals at high speed, generating step pulses for the stepper motor, or quickly reacting to a limit switch trigger can be handled by the Teensy's firmware. The Jetson can offload low-level control to the Teensy. One scenario: the Jetson might decide "move weeder down 5 cm and then rotate it," and it would send a command to the Teensy; the Teensy would then produce the exact sequence of stepper pulses to move the stepper motor the correct number of steps, monitor the limit switch in case it triggers unexpectedly, and perhaps control the DC motor's PWM for rotation – all in a tightly timed manner. The Teensy, with its 600 MHz speed, can manage multiple such tasks in parallel (it has multiple timers and DMA channels for handling I/O). Additionally, the Teensy can preprocess sensor data – for example, integrating wheel encoder counts and streaming the odometry to the Jetson in a convenient format. Because it runs as a microcontroller, it can attain true real-time performance, meaning it can

guarantee responses within a few microseconds or milliseconds, something the Jetson's OS cannot always do due to multitasking.

In summary, the Teensy 4.1 is like the robot's motor and sensor coordinator. It interfaces with the hardware (motors, encoders, switches) and ensures those components act timely according to the high-level commands. It might also handle communication with the ODrive motor driver (via CAN or serial) on behalf of the Jetson, packaging commands and forwarding telemetry. Its ARM Cortex-M7 processor brings "desktop-class" processing to microcontroller tasks, giving plenty of headroom for handling complex math (like control algorithms) quickly [59]. Thus, the Teensy 4.1 significantly contributes to the robot's ability to react in real time, providing a bridge between the Jetson's high-level decisions and the physical actions carried out by motors.

4.4.2 ODrive v3.6 (High-Power Motor Controller)

What it is: The ODrive v3.6 is a dual-channel electronic motor controller specifically designed to drive brushless DC motors with encoder feedback, effectively turning them into servo motors. ODrive is an open-source project aimed at affordable high-performance motor control. The board can control two motors simultaneously, handling substantial current (up to 120A peaks per motor) with supply voltages up to 24V or 56V depending on the version [60]. It has built-in microcontroller firmware that implements field-oriented control (FOC) for smooth torque control of brushless motors, and it interfaces with quadrature encoders to close the loop on position/velocity [61]. Essentially, instead of buying industrial servo drives, the ODrive lets makers use hobby brushless motors (like those found in e-bikes or drones) as precise servos. It supports various control modes (position control, velocity control, torque control) and can be commanded via several interfaces: USB, UART, or CAN. In this robot, ODrive likely runs configured so that you can tell it "motor1 at X radians/second" or "motor1 go to position Y" and it will manage the currents and voltages to the motor phases, using the encoder to ensure it achieves the target.

Role in the robot: The ODrive is the motor driver for the locomotion wheels (and possibly any other high-power motors like a mowing blade if brushless). The two drive wheels of the mower are probably each attached to a brushless

DC motor, and the ODrive v3.6 controls both. Its job is to take motion commands (from the Teensy or Jetson) and output the heavy currents needed by the motors, all while precisely monitoring motor rotation via encoders. This relieves the rest of the system from the burden of direct motor control. For example, if the robot needs to drive forward, the Jetson/Teensy might send a velocity command to ODrive: "left wheel 1.0 m/s, right wheel 1.0 m/s." The ODrive will then ramp up the voltage to the motors, measure the encoder speeds, and adjust power dynamically to maintain those speeds. It will correct for disturbances (like thicker grass causing drag) almost instantaneously by increasing motor current, acting as the low-level servo controller for the wheels. Without the ODrive, implementing such fine motor control would be a complex task for the Teensy or Jetson; with ODrive, it's largely plug-and-play – you tune the controller once and then stream high-level commands to it. Additionally, ODrive provides feedback like motor currents, motor position, etc., which can be used for monitoring the health of the drive system (for instance, detecting if a wheel is stuck because current spiked).

In essence, ODrive v3.6 is to the drive motors what a car's ECU is to the engine – it directly operates the motors using feedback control to meet the requested performance. This means the robot's wheels behave like true servos with precision and power: they can hold a fixed speed or position under load. The inclusion of ODrive demonstrates the design's emphasis on robust locomotion; it leverages an existing high-power controller so that the team doesn't have to build one from scratch. Overall, ODrive ensures the two main drive motors work in unison and respond accurately, which is critical for straight-line mowing and for gentle handling (e.g., slowing down when approaching an obstacle).

4.5 Communication Methods: Networking the Robot's Components

Inside the robot, multiple electronic components need to exchange information: the Jetson computer issues commands and needs sensor data, the microcontrollers coordinate with each other, and the robot might even be accessed from an external PC for updates or monitoring. To facilitate this, the system uses several communication protocols, each suited to specific needs. The main communication methods in this design are Serial communication, CAN Bus,

and Peer-to-Peer (P2P) SSH connections. Using multiple methods is necessary because different parts of the system have different requirements (speed, reliability, topology). Below we explain each:

4.5.1 Serial Communication (UART)

What it is: Serial communication refers here to the standard asynchronous serial interface (UART/USART), which sends data one bit at a time over a line (plus a line for reception in the opposite direction). This is the same kind of communication used by Arduino's USB connection (via a USB-serial chip) or the typical TX/RX pins on microcontrollers. In serial communication, bytes of data are framed with start and stop bits and sent sequentially. It typically involves two devices: one is the transmitter, the other the receiver, though the roles can swap. Because it uses only a couple of wires, it's simple and low-cost. Common serial formats in robots include TTL-level UART (e.g., 5V or 3.3V signals between MCU boards) and RS-232 or USB serial to a PC. Baud rates can range from a few thousand (9600 bps) to millions of bits per second, depending on the devices and wiring.

Use in the robot: The lawn robot likely uses serial communication for straightforward board-to-board links. For example, the Arduino Mega may be connected to the Jetson Orin Nano via a USB cable acting as a serial port. The Jetson can send text or binary commands to the Arduino (e.g., "REPORT BATTERY" or motor on/off signals) and the Arduino can send sensor readings or status messages back. Similarly, the Teensy 4.1 might use a UART to communicate with the Jetson or with the ODrive if CAN is not used. Serial is used because it is simple and universally supported – virtually all microcontrollers and SoCs have UART capability, and it's easy to implement in software. It's suitable for moderate-speed data exchange and commands, such as sending a few readings per second or issuing high-level instructions.

The reason to include serial despite having CAN is that some subsystems might not support CAN or don't need its complexity. Also, during debugging or development, a serial link is invaluable (for logging to a console, etc.). One could plug a laptop into the robot's Arduino Mega via USB and read debug messages. In operation, serial lines could be used for the Arduino Mega to send the Teensy the measured battery voltage (as a simple number), or for the

Jetson to tell the Arduino to activate a buzzer or lights. In summary, serial communication in this robot is the straightforward, point-to-point channel for low-bandwidth, non-time-critical communication. It's the simplest glue linking controllers at a high level, leveraging the fact that "serial communication requires fewer lines or wires" and is very easy to use [62]. Every distinct pair of devices that need a direct chat can use a serial link if appropriate.

4.5.2 CAN Bus (Controller Area Network)

What it is: CAN bus is a robust vehicle networking protocol designed originally for cars to allow microcontrollers and devices (ECUs – Electronic Control Units) to communicate over a shared two-wire bus [63]. Unlike serial, which is typically just one-to-one, CAN is multi-drop: many devices can be connected to the same two wires (CAN High and CAN Low). They send messages in a prioritized manner – if two devices talk at once, the one with higher priority ID wins the bus and the other backs off. CAN frames are short (up to 8 data bytes in classic CAN) and include error checking. It's known for reliability in noisy environments (the differential signaling and robust error detection make it resilient to electromagnetic noise). In a car or robot, you can think of CAN as a nervous system, where each node (e.g., motor controller, sensor unit) broadcasts information that all other nodes can listen to [63]. For example, one node can broadcast "Wheel Speed = 1.2 m/s" and multiple controllers could use that data. CAN bus typically runs at up to 1 Mbit/s in classic CAN (and higher in newer CAN-FD), which is fast enough for real-time sensor/actuator data in a robot.

Use in the robot: In this autonomous lawn mower, CAN bus is likely used to connect the Jetson (or Teensy) with the ODrive motor controller, and possibly with the BMS or any other smart actuators. For instance, ODrive supports CAN, so both drive motors could be commanded over a single CAN bus. The Teensy or Jetson can send a CAN message like "ID 0x001: left motor target speed = X" and "ID 0x002: right motor target speed = Y", and the ODrive will respond on the same bus with messages like "ID 0x101: left motor actual speed = V, current = I". The advantage here is that only two wires connect the Jetson/Teensy and ODrive, and this single bus could extend to other devices too. If the battery's BMS is a smart one with CAN output (as used in some electric vehicles or robots), it could put messages on the CAN bus about battery

status ("ID 0x200: battery voltage = ..., current = ..."), which both the Jetson and motor controller could listen to.

The use of CAN shows an emphasis on reliable, multi-device communication. In a high-electrical-noise environment like a lawn mower (where motors and cutting blades can cause voltage spikes and interference), CAN's robustness ensures data gets through intact or not at all (with automatic retransmission on errors). Also, CAN being multi-master means there's no single point of failure for communication; any node can initiate a message when needed. This suits a distributed control system – e.g., the Teensy could autonomously broadcast wheel encoder data on CAN at high rate, while the Jetson listens and also sends steering commands. All of this can happen concurrently on the bus.

In summary, CAN bus is the robot's backbone network for critical real-time data between controllers and smart devices. It is used because of its noise immunity, efficiency in handling multiple devices, and priority-based messaging which is ideal to ensure, say, emergency stop messages get bus priority over less urgent data. Many modern robots adopt CAN for these reasons, as it "enables quick and reliable sharing of information" in a system with many microcontrollers [63].

4.5.3 P2P SSH (Peer-to-Peer Secure Shell)

What it is: SSH stands for Secure Shell, a network protocol that allows secure remote login and command execution on a device. Typically, one might use SSH over a standard network (Ethernet/Wi-Fi) to connect to the Jetson's Linux terminal from a laptop. "Peer-to-peer SSH" in this context likely means setting up an SSH connection directly to the robot over an ad-hoc or local network, possibly even through a dedicated wireless link or a hotspot, without going through any cloud server. It could involve the robot acting as a Wi-Fi access point to which a user connects and then SSHes into the robot's onboard computer. P2P implies a direct connection between two peers (the user's device and the robot) rather than through an intermediary. SSH provides an encrypted channel, meaning even though it's wireless, the commands and data are protected.

Use in the robot: P2P SSH is used for remote communication with the robot's Jetson for maintenance, monitoring, and control. While the robot is

meant to operate autonomously, during development and possibly for advanced users, it's extremely useful to be able to connect to the robot and see what's going on or to intervene. Through an SSH session, a developer (or the thesis student) can log into the Jetson Orin Nano from their laptop to do things like: start or stop ROS nodes, adjust configuration files, upload new code, or check system logs and diagnostics in real time as the robot is running. Since Jetson runs Ubuntu, SSH is a natural way to get a Linux shell remotely. If the robot is mowing the lawn and something seems off, one could SSH in over Wi-Fi and observe sensor outputs or CPU usage. The term "peer-to-peer" suggests that the robot might not be on a home Wi-Fi network but rather directly connecting, which is useful in a big yard or in the field without router coverage. Possibly the robot might have a cellular or other peer-to-peer link as well, but more likely it's Wi-Fi direct.

In terms of architecture, this is not a bus or internal comm link like the others, but rather a method for an external user or external computer to communicate with the robot. It's worth noting that SSH is also used for file transfer (SCP/SFTP) and tunneling, so the user can update software securely. Additionally, if the robot needed to send data to a base station (like sending an alert "I am stuck" or telemetry), an SSH-based connection could be used in reverse (the robot SSHes into a server or opens a reverse tunnel). The key point is, SSH provides a secure and reliable way to interface with the robot's brain from the outside world. It is strongly authenticated and encrypted, so only authorized persons can access the robot. This is important because the robot is powerful and has spinning blades – you want to ensure no one can hijack it. So the use of SSH aligns with maintaining security while enabling convenient remote access.

In summary, P2P SSH doesn't control motors directly or carry sensor data in normal operation; instead, it's the development and maintenance lifeline. It allows the humans behind the robot to peer inside the system, update it, or manually drive it (for example, one could SSH in and manually publish ROS commands to move the robot, essentially teleoperating it from a computer). Using SSH means the team can avoid attaching screens or keyboards to the robot – everything can be done wirelessly through a terminal. This greatly speeds up development and debugging and will be a valuable tool for thesis experiments and demonstrations.

4.6 Robot Operating System (ROS): Software Integration Framework

Finally, it's worth mentioning the role of Robot Operating System (ROS) in this system. ROS is not a physical component but a software framework that likely underpins how all these components interact at the software level on the Jetson (and possibly on the Teensy/Arduino via ROS-compatible firmware or bridging nodes).

What ROS is: The Robot Operating System (ROS) is an open-source middleware and toolset for robotics software development. It provides a structured communications layer above the operating system – with concepts like nodes (independent software modules), topics (named buses over which nodes publish/-subscribe messages), and services/actions (for request-response or longer goals) [64]. ROS is often described as a "flexible framework for writing robot software" that offers hardware abstraction and message-passing between processes [64]. This means a developer can write one module for processing camera images, another for planning paths, another for controlling motors, and ROS handles the exchange of data between them in a standardized way. ROS also comes with a vast array of pre-written packages (for SLAM, for sensor drivers, for visualization, etc.) which can be reused.

Significance in this robot (integration): In the autonomous lawn robot, ROS likely serves as the integration backbone for all subsystems. For example, there might be a ROS node that interfaces with LIDAR (reading data from the LIDAR driver and publishing a LaserScan or point cloud message), another ROS node for the stereo camera (publishing image streams and perhaps depth images), and another node that takes all this sensor data and runs an AI model to detect weeds. The Jetson would run these nodes concurrently. ROS allows each part to be developed and tested somewhat in isolation, with clear interfaces (messages). The navigation stack (perhaps using ROS Navigation or a custom strategy) can subscribe to sensor topics and output velocity commands to a /cmd_vel topic. A motor control node (connected to Teensy or ODrive) would subscribe to /cmd_vel and then send the appropriate low-level commands to the motors. Meanwhile, the encoders might be publishing an /odom topic (odometry), which feeds into the navigation stack for localization. All of this

is facilitated by ROS: rather than writing one giant program, the software is modular.

Using ROS also provides valuable tools during development – for instance, RViz (ROS Visualization) to visualize the robot's perception (point clouds, camera images, detected weed positions, etc.) and rosbag to record data for analysis. It helps in tuning algorithms because you can replay sensor data and adjust parameters. ROS also takes care of message queuing, so if one part of the system runs slower, messages buffer up rather than being lost (within limits). Essentially, ROS acts as the central coordinator in software, making sure all these complex components (sensors, actuators, decision-making nodes) can communicate seamlessly. It is often called the operating system of the robot (though it runs on top of Linux) because it provides many services you'd expect: hardware abstraction, low-level device control, implementation of common functionality, message passing between processes, etc [64].

By using ROS, the developers avoid reinventing the wheel for common functionality. They can use, for example, a ROS driver for the LIDAR, a ROS node to talk to the ODrive (there are existing implementations for ODrive and Teensy to ROS), and leverage community-tested libraries. It also means the system is easier to extend or modify. If, say, a new sensor is added (like a GPS), one can integrate it by adding a node without overhauling the entire code base – just publish new info and perhaps subscribe where needed. In summary, ROS is the software glue and architecture that binds together the sensing, thinking, and acting parts of the robot. It ensures that the output of one component (say, the LIDAR distance readings) can become input for another component (obstacle avoidance planner) with minimal custom coding, thanks to standardized messages. Moreover, ROS encourages a clean separation of concerns (each node has its job), which aligns well with the hardware modularity (each sensor/controller has its own role as described above). Thus, while navigation and AI algorithms are beyond the scope of this hardware-focused chapter, ROS is mentioned as a critical enabler that "provides the services that a typical operating system offers, such as hardware abstraction, low-level device control, message-passing between processes, and package management" [64], all of which are used to integrate the lawn robot's many components into one coherent autonomous system.

Concluding Remarks: Through this array of components – from laser scanners and cameras giving the robot sight, to motors and controllers giving it motion, all coordinated by advanced computing and communication – the autonomous lawn maintenance robot is equipped to perform its tasks intelligently. The sensors serve as the robot's eyes, providing rich data about the environment and its own state; the actuators are its hands and tools, physically interacting with the lawn; the Jetson Orin Nano SoM is the brain that thinks and makes decisions, heavily utilizing AI; microcontroller-based controllers like the Teensy and Arduino act as the nerves and reflexes, ensuring smooth and timely execution of motions; communication networks (Serial, CAN) are the nerves wiring everything together; and ROS is the operating framework that integrates sensing, decision, and actuation into a harmonious whole. By understanding each component's function and role, we can appreciate how they work in concert to achieve complete autonomous lawn care – safely and efficiently mowing grass, removing weeds, and edging boundaries, all without human intervention. This modular yet integrated design allows each subsystem to do what it's best at, resulting in a robust overall system as detailed in this chapter.

4.7 Summary

This chapter outlines the key hardware and communication architecture of the autonomous lawn care robot. It begins with the sensors, which include LIDAR, stereo cameras, limit switches, encoders, and voltage sensors. These serve as the robot's visual and feedback systems, enabling environmental awareness, safety, and precise movement. Actuators are then discussed, with servo motors used for drive control, stepper motors for positioning the weeder, and brushed DC motors for rotational weed removal. The NVIDIA Jetson Orin Nano is introduced as the robot's central processing unit, running AI tasks and managing perception and navigation through the Robot Operating System.

To control hardware in real time, microcontrollers such as the Teensy 4.1 and Arduino Mega handle sensor inputs and motion commands, while the ODrive v3.6 manages brushless motor operation. Communication between these systems is supported through Serial, CAN Bus, and SSH, ensuring coordination between internal components and remote access for updates and diagnostics. These

interconnected modules enable the robot to perform autonomous mowing and weeding reliably. The chapter highlights how each component contributes to a scalable and intelligent system for modern lawn care.

Chapter 5

Object Segmentation, Tracking and Integration

The perception system is a critical part of the robot, responsible for detecting target objects in real-time. The original goal was to build a custom neural network to detect dandelion weeds in a field. However, challenges were encountered in collecting a sufficient and varied weed image dataset (dandelions are seasonal and hard to capture consistently). To keep the project moving forward, a pivot was made to use a more controllable object as the target for detection. Wheels were chosen as the stand-in target object. Wheels are readily available and easy to photograph, providing a reasonable proxy to develop and test the vision pipeline. The dataset included two distinct types of wheels to simulate a multi-class scenario. A YOLOv8 object detection model was then trained on this wheel dataset. The model can detect wheels in real time (about 25 frames per second) with high accuracy (around 97% detection accuracy on the test data). This section details the dataset collection and annotation process, the neural network training, the real-time inference and tracking method, and how the perception output is integrated with the robot's controller for navigation.

5.1 Dataset Collection and Annotation

To begin, a small dataset using two different RC tyre wheels was created. The wheels, in various positions and angles, were captured by an Intel Realsense camera to take pictures and videos. This setup provided both RGB and depth

information, although the depth data was not used. The dataset preparation process also included ensuring variation by capturing wheels on different backgrounds, at varying distances, and with partial occlusions in some cases. This approach allowed the model to learn to recognize the wheels in diverse situations. In total, approximately 500 images were gathered, with around 250 images per wheel type. This number was deemed sufficient for a prototype model, given the simplicity of the object shapes.

After collecting the raw images, the annotation process began. The annotation involved drawing bounding boxes around each wheel in the images and labeling them with the correct class, either "Wheel Type 1" or "Wheel Type 2." The annotation was performed using an open-source tool called LabelImg. Each image was processed by carefully drawing a rectangle around the wheel and assigning the appropriate label. This step was time-consuming but essential for training an accurate detector. Another annotating software, Roboflow, was also explored during the course of the labeling datasets. Different ways of labeling datasets are shown in Figure 5.1 and 5.2.

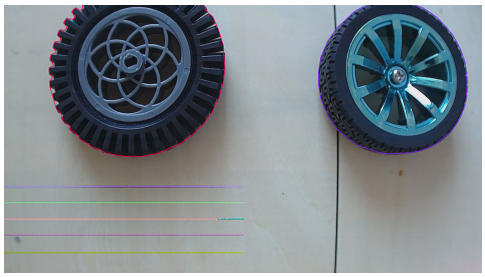

Figure 5.1: Before smart labeling

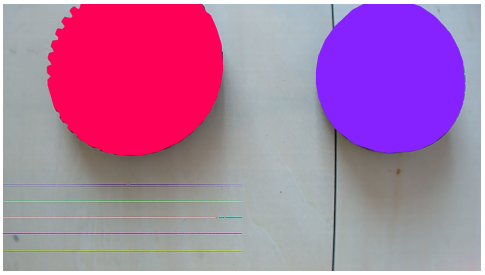

Figure 5.2: After smart labeling

The annotations were saved in YOLO format. For each image, a corresponding text file was generated, listing the bounding box coordinates and the class of each wheel in the image. The YOLO format includes the class ID, along with the normalized x and y coordinates of the box center, as well as its width and height relative to the image size. This format was chosen because YOLOv8 natively accepts it, simplifying the training process. During annotation, extra care was taken to ensure that both wheel types were well-represented and that no labeling errors occurred, such as a bicycle wheel being incorrectly labeled as a car wheel. This meticulous dataset preparation formed a solid foundation for training the neural network.

5.2 Training an AI Neural Network Model

With the annotated wheel dataset prepared, training of the object detection model began. YOLOv8 was selected for the detection architecture. YOLOv8 (You Only Look Once, version 8) is a state-of-the-art convolutional neural network model for object detection. YOLOv8 was chosen due to its speed and accuracy, which are crucial for real-time robot vision. Another advantage is that YOLOv8 provides convenient training workflows and pre-trained weights. A YOLOv8 model pre-trained on a large dataset (the COCO dataset) was fine-tuned on the wheel dataset. By leveraging a pre-trained model, the training time and data requirements were reduced, as the model had already learned general features and only needed to be taught to recognize the specific wheel classes. The training environment was set up on a machine with a GPU to speed up the process (an NVIDIA RTX-series graphics card on a desktop PC was used; alternatively, Google Colab with a Tesla T4 GPU was used for some training runs). The training was done using the Ultralytics YOLOv8 library in Python, which facilitated easy configuration and execution. The model was trained for 100 epochs, making 100 passes through the entire wheel dataset. A batch size of 16 images per iteration was chosen, balancing GPU memory usage and stable training updates. During training, the model used the annotated images to gradually learn the features of each wheel type. Data augmentation techniques built into YOLOv8 (such as random flips, scales, and color adjustments on the images) were enabled to improve the model's ability to generalize. This allowed the model to see slightly varied versions of the wheels, improving its robustness to real-world variations.

Over the epochs, training loss and validation metrics were monitored. A small portion of the data (about 10% of the images) was set aside as a validation set to evaluate the model at each epoch, helping prevent overfitting. The training process took several hours. By the end, the model's performance was quite good. On the validation data and an additional test set (another 10% of images that the model had never seen during training), the wheel detector achieved roughly 97% accuracy. In practical terms, this meant the model correctly identified and localized wheels in 97 out of 100 cases. The result was considered very satisfactory, as it indicated the model learned to recognize both types of wheels with high precision. It was also verified that the model was distinguishing the two classes correctly (for example, labeling the bicycle wheel vs. the car

wheel appropriately), with only a few minor confusions on a couple of images. Importantly, the inference speed of the model was excellent: around 25 Frames Per Second (FPS) when running on the GPU. This speed fulfilled the real-time requirement (a typical video runs at 24–30 FPS), allowing the model to process camera frames virtually as they arrived. Even on a CPU (such as running on a laptop without the GPU), a decent frame rate was achieved (though lower, around 8–10 FPS). Since the robot's perception was to be run on an onboard computer with a GPU, 25 FPS real-time was considered achievable. After training, the best model weights (a .pt file with the learned parameters) were exported for use in the inference stage on the robot.

5.3 Inference and Tracking

Training the model was only half the task – deploying it for inference, or running the model on live video feed to detect wheels in real-time, was the next step. A simple inference script was written to process frames from the robot's camera using the trained YOLOv8 model. The robot is equipped with a forward-facing camera, and the script captures each frame (image) from this camera (using OpenCV for video capture). For each frame, the image is passed through the YOLOv8 model, which outputs any detected objects (in this case, the coordinates of the bounding box around a detected wheel and the class label of the wheel type). A confidence threshold was set so the model only reported a detection if it was reasonably sure (e.g., above 0.5 confidence). During tests, as soon as a wheel appeared in the camera's view, a bounding box was drawn around it by YOLOv8, typically with minimal delay. Detection felt instantaneous, thanks to the 25 FPS processing capability – effectively analyzing every frame in real-time. However, detecting objects in each frame is not always sufficient for a smooth perception system on a moving robot. When the robot or object moves, the detection can sometimes jitter (the bounding box may shift or occasionally a frame may miss the object if the view is blurry). To address this, an object tracking mechanism was integrated into the perception system. The goal of tracking is to maintain the identity and position of the detected object consistently across frames. YOLOv8's built-in tracking functionality was used, which relies on a tracker like ByteTrack. In practice, when the wheel is first detected, it is assigned an ID (e.g., ID #1). On subsequent frames, the tracker looks at the new detections and the previous

position of ID #1. If the same wheel is still in view, the tracker keeps the ID consistent and updates the position.

Figure 5.3: Inference and Tracking

This way, the system knows it's the same object moving, rather than interpreting a new wheel as appearing every frame. The tracking component significantly improved the stability of perception. For example, if the robot was moving slightly and the camera feed shook, the detector might lose the wheel for one frame. With tracking, the system predicted the wheel's position and maintained its identity until the detector picked it up again. Tracking was also useful when multiple objects were in view. In one experiment, two wheels (one of each type) were placed in front of the robot. The YOLOv8 model detected both, and the tracker assigned them unique IDs. As one wheel moved, the tracker could follow it specifically, which is helpful for the controller to decide which object to pursue. The tracking ran at the same speed as detection, with minimal overhead, maintaining the 25 FPS rate overall. At the end of this stage, a perception pipeline was established that could detect wheels in real-time and steadily track their positions from frame to frame. On a screen, this was visualized as a box that smoothly followed the wheel, rather than jumping around. This indicated that the robot would have a reliable stream of information about the target object's location. An example of segmentation and tracking can be seen in Figure 5.3.

5.4 Integration of Inference and Tracking to the Controller

With real-time detection and tracking functioning, the final step was to integrate the perception system with the robot's controller. The robot's controller is the component that drives the wheels and makes navigation decisions. In this setup, a separate microcontroller handles motor control for movement, and an onboard computer (such as a Raspberry Pi or NVIDIA Jetson) runs the vision system. A communication link between the two systems was required for the vision system (running YOLOv8 and tracking) to send information to the motion controller so the robot could react to a detected wheel. A simple communication link was implemented, using a serial connection (USB) to send data from the laptop/Jetson running the inference to an Arduino-based motor controller on the robot. Each time the YOLOv8 model detected a wheel, relevant information (such as the position of the wheel in the image frame) was packaged into a short message. Specifically, the center of the wheel's bounding box was calculated and normalized with respect to the image width. This offset value indicated how far left or right the wheel was relative to the center of the camera's view. This offset was sent over the serial link to the robot's controller, at a rate of approximately 10 times per second. On the microcontroller side, code was written to read the incoming messages and adjust the robot's movement accordingly. The integration worked as follows: if the wheel was centered in front of the robot, the controller commanded the robot to drive straight. If the wheel moved to the left side of the camera frame, the controller turned the robot left to re-center it, and similarly for the right side. This created a basic visual servoing system where the robot steered itself based on the visual target's location, continuously correcting to keep the wheel in the center of view. The tracking component augmented this behavior by ensuring that, even if the detection temporarily failed, the last known position was used to guide the robot, preventing erratic behavior. If the wheel went out of frame (indicating it may have been reached or passed), the robot would stop or search for it again. This perception-to-control feedback loop greatly enhanced the robot's navigation capabilities. In testing, placing a wheel on the ground allowed the robot to detect, track, and follow the wheel as it moved. The integration process structure is shown in Figure 5.4.

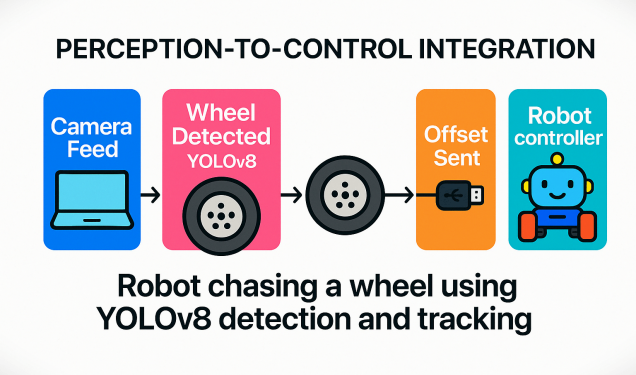

Figure 5.4: Workflow of Integration

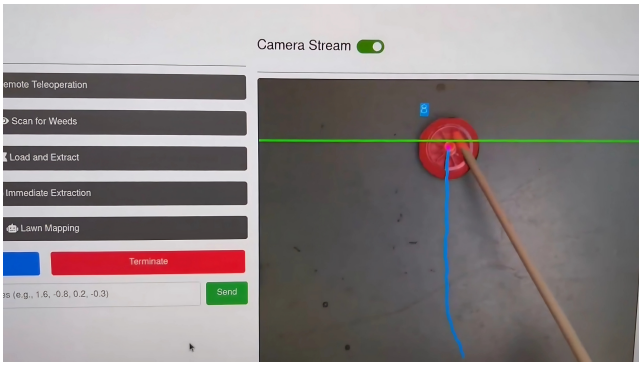

Figure 5.5: ID Number sent to Controller

As a result, the robot was able to drive toward the wheel and follow it when moved. This is analogous to the desired future goal for weed detection: the robot would detect a weed and navigate to it for removal. As soon as the object enters the green line as shown in Figure 5.5 the ID number with its position is sent to the controller for the robot to be able to navigate to that point. Although demonstrated with wheels, the integration is designed to be general, allowing the same approach to be applied to dandelion weeds in the future. The smooth communication between the vision system and the controller meant the robot responded in near real-time to what the camera saw. The latency from detection to movement was minimal (on the order of tens of milliseconds), which is negligible for practical purposes. Overall, this integration demonstrated that the perception system could effectively drive the robot's actions. The robot's intelligent response to the visual target provided tangible validation of the entire perception pipeline.

5.5 Summary

This chapter describes the development and implementation of a vision-based perception system that allows the robot to detect and track target objects in real time. Due to the limited availability of dandelion images, RC wheels were used as proxy targets. A custom dataset featuring two types of wheels was captured using an Intel RealSense camera and annotated with tools like LabelImg and Roboflow to produce YOLO-compatible bounding box labels.

A YOLOv8 model was trained on this dataset, achieving around 97 percent accuracy and running at approximately 25 frames per second on a GPU. ByteTrack was used to ensure stable tracking with unique IDs across frames. The system transmitted bounding box offsets to the Arduino-based controller via serial communication, allowing the robot to adjust its movement and center itself on the target. This created a closed-loop system that allowed the robot to follow a moving target, validating a full perception-to-action pipeline that is modular and adaptable for future weed detection and removal tasks.

Chapter 6

Kinematic Control of Differential Drive

This chapter presents the complete control architecture implemented for Dandebot, an autonomous differential-drive robot designed for precise lawn maintenance operations. The control stack spans from fundamental kinematic modeling and MATLAB-based simulation to hardware-level PID motor control and ROS 2-based high-level navigation. The chapter begins by analyzing the robot's motion through forward and inverse kinematics, then validates basic motion through simulation using MATLAB. It then transitions to the real-time motor control strategy using cascaded PID loops in ODrive, followed by high-level decision-making and trajectory tracking implemented via ROS 2's Dynamic Window Approach (DWB) controller. Together, these layers enable the robot to safely and efficiently navigate toward task-specific targets like weeds or mowing zones.

6.1 Forward Kinematic Motion Analysis

In this section, the mathematical model of Dandebot's motion is derived based on its differential-drive configuration. The forward kinematic equations describe how wheel velocities translate into global position and orientation changes over time. This foundational model forms the basis for later simulation and control design. The section also includes a MATLAB implementation to visualize how the robot responds to constant wheel speeds.

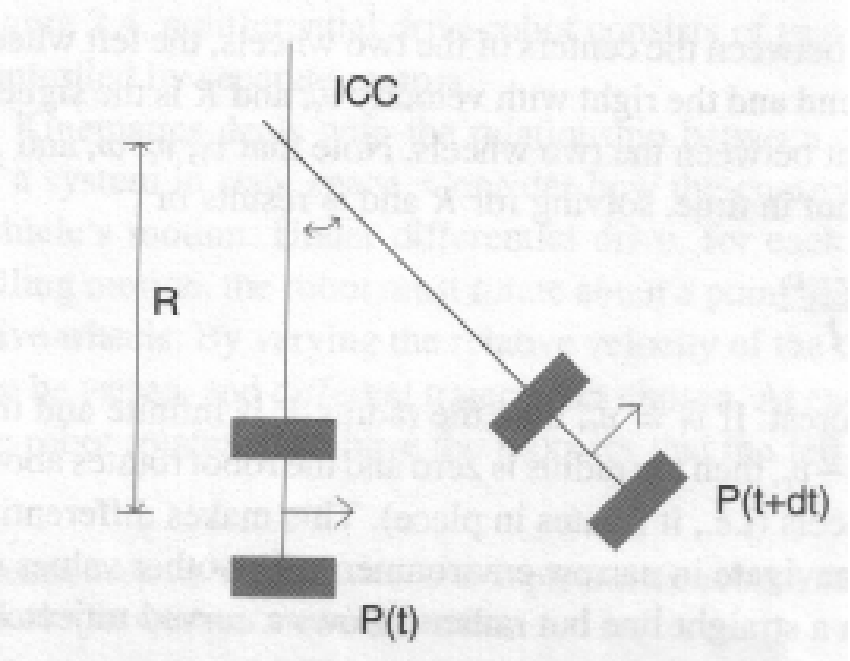

Figure 6.1: Differential drive kinematics [65]

Before trying MATLAB, the theoretical part was deeply studied. Now, assuming the robot is at some position (x,y), headed in a direction making an angle θ with the X axis. We also assume the robot is centered at a point midway along the wheel axle. By manipulating the control paramters V_l, V_r, we can get the robot to move to different positions and orientations. (note: V_l, V_r) are wheel velocities along the ground.

Knowing velocities and using equation that we derived in the previous section:

$$ICC = [x - R\sin(\theta),\ y + R\cos(\theta)] \tag{6.1}$$

and at time $t + \delta t$ the robot's pose will be:

$$\begin{bmatrix} Y_x \\ Y_y \\ Y_\theta \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} \tag{6.2}$$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} Y_x \\ Y_y \\ Y_\theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{6.3}$$

The above equation simply describes the motion of a robot rotating a distance R about its ICC with an angular velocity of ω [65]. Referring to Figure 6.1, another way to understand this is that the motion of the robot is equivalent:

1) Translating the ICC to the origin of the coordinate system
2) Rotating about origin by an angular among $\omega\delta t$
3) Translating back to the ICC.

To be able to understand what is behind the kinematic model and how those parameters actually change, we simply try to visualize them in MATLAB. To begin with we proceed with Forward Kinematics.

- *Definition:* Forward kinematics involves determining the position and orientation of the mobile robot given the motor joint parameters.
- *Parameters:* The parameters in this simulation are mainly ω_l and ω_r which are denoted by $omega_1$ and $omega_2$ in the below matlab code. The whole code can be referred in APPENDIX A.

```
1  eta0 = [x0;y0;psi0];
2  eta(:,1) = eta0;
3  omega1=1; omega2=0.5;
4  for i=1:length(t)
5      psi=eta(3,i);
6      J_psi=[cos(psi),-sin(psi),0;
7             sin(psi),cos(psi),0;
8             0,0,1];
9        W=[a/2,a/2;
10          0,0;
11          (-a/(2*d)),(a/(2*d))];
12       omega=[omega1;omega2];
```

```
13      zeta(:,i)=W*omega;
14      eta_dot(:,i)=J_psi*zeta(:,i);
15      eta(:,i+1)=eta(:,i)+dt*eta_dot(:,i);
16  end
```

Listing 6.1: Forward Kinematic Motion

This initializes the robot's state eta, where x0, y0, and psi0 are its initial x-coordinate, y-coordinate, and orientation (angle in radians), respectively. We also set constant wheel velocities (omega1 and omega2) for the robot's left and right wheels, respectively.

A loop over all time points defined in the vector t begins and updates the robot's state for each discrete time step. Here, the robot's current orientation psi is extracted from its state. Then, a transformation matrix Jpsi is created based on this orientation. This matrix is used to convert velocities in the robot's local frame to velocities in a global (or fixed) frame. The matrix W relates wheel velocities to the robot's linear and angular velocities. The values a and d might refer to certain robot parameters, such as its width or wheel radius. Multiplying W with the wheel velocities (omega) gives the robot's linear and angular velocities in its local frame. Then it calculates the robot's velocity in the global frame by transforming its local frame velocities using the matrix Jpsi.The robot's state is updated using a simple Euler integration. The robot's velocities (etadot) are multiplied by a time step dt and then added to its current state to compute its next state. The Results can be seen below in the Figures 6.2 and 6.3. Figure 6.2 depicts the robot's position at time t_0 and the Figure 6.3 depicts the robot's position at time t_{25}.

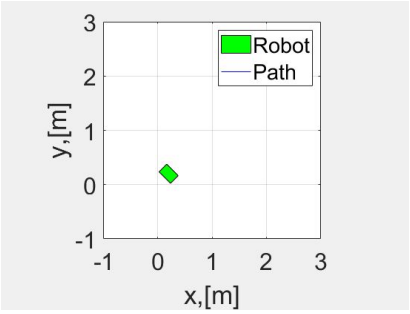

Figure 6.2: Robot's Position at t_0

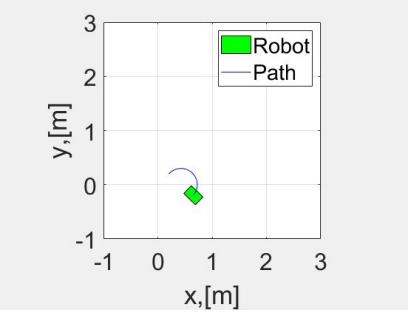

Figure 6.3: Robot's Position at t_{25}

6.2 Inverse Kinematic Control in Polar Coordinates

This section introduces inverse kinematics by expressing the robot's movement toward a goal in polar coordinates. By transforming the pose error into distance and angular components (ρ, α, β), we define control laws that guide the robot to a target while maintaining smooth orientation transitions. MATLAB simulations demonstrate how the robot follows a goal point using this control strategy.

Next, we can move with Kinematic Control using Polar Coordinates. We will try to analyse the change in states using visualization tools provided by MATLAB. When controlling mobile robots, one of the effective strategies is to convert Cartesian errors (those in the $x, yand\theta$ space) into polar errors, particularly when navigating towards a target location. This is achieved by translating the state errors in Cartesian space into polar coordinates ρ, α, β.

6.2.1 How Polar Kinematic Controls Work

1. Conversion to Polar Coordinates:
    - The robot's position error in Cartesian space x, y relative to a goal position x_g, y_g is first transformed into polar coordinates.
    - ρ represents the distance between the robot and the target.
    - α represents the angle to the target from the robot's current orientation.
    - β is the bearing angle to the target from the robot's current heading.
2. Error Consideration:
    - In polar kinematics control ρ, α, β are the primary errors being considered.
    - ρ determines how far the robot is from its target.
    - α and β provide insights into the robot's orientation relative to its target.

That was a reference to the theory explained in last chapter. The purpose of the linear control law is to reduce the error.

- The robot moves in such a way that it tries to reduce the distance ρ to the goal, thereby reaching closer to the destination.
- The control law for angular velocity ensures that the robot maintains an orientation that minimizes α and β, making the approach to the target more efficient.

```
gc(:,1) = [x0;y0;psi0];
Kp = 3;
Ka = 8;
Kb = -1.5;

eta = zeros(3, length(t));
for i=1:length(t)
    delx = xd-gc(1,i);
    dely = yd-gc(2,i);
    psi = gc(3,i)
    rho = sqrt(delx^2+dely^2);
    alpha = -psi + atan2(dely,delx);
    beta = - psi - alpha ;
    eta(:,i) = [rho;alpha;beta];
    J = [-cos(alpha),0;(sin(alpha)/rho),-1;(-sin(alpha)/rho),0];
    zeta = [Kp*rho;Ka*alpha+Kb*beta];
    eta_dot(:,i) = [-Kp,0,0;0,-(Ka-Kp),-Kb;0,-Kp,0]*eta(:,i);
    eta(:,i+1) = eta(:,i) + eta_dot(:,i)*dt;
```

Listing 6.2: Inverse Kinematic Polar Coordinates Control

The provided MATLAB code implements a control strategy for a differential drive robot based on its polar coordinates. the script initializes the robot's starting position and orientation. The gains K_p, K_a, K_b are the tuning parameters for the control law.

From this code, it's clear that the control strategy aims to drive the robot to a desired position xd, yd by computing the control inputs based on the polar coordinates. The control law uses the gains $Kp, Ka, andKb$ to ensure that the robot reaches its target as shown in Figure 6.5 and 6.6 while minimizing errors in the polar coordinates. The x, y position plot from t_0 to t_5 can be seen in Figure 6.7.

The breakdown of the control loop can be seen below in Figure 6.4:

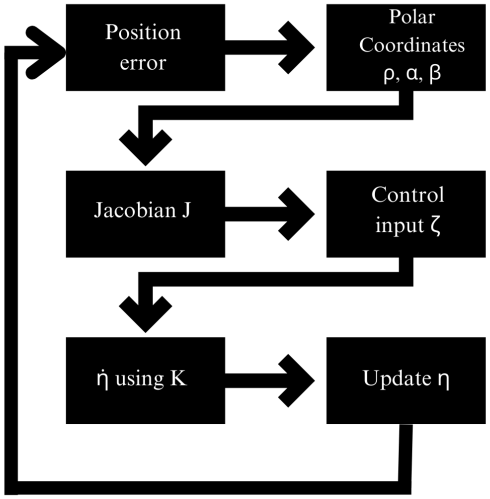

Figure 6.4: Control Loop Breakdown

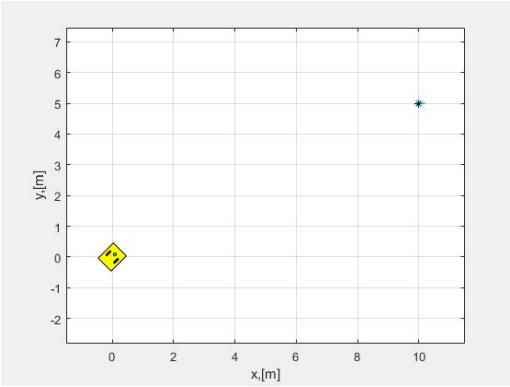

Figure 6.5: Robot's Position at t_0

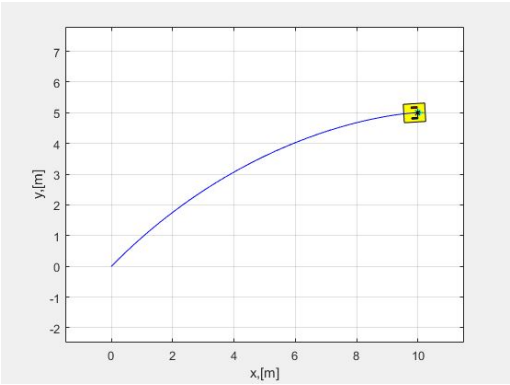

Figure 6.6: Robot's Position at t_5

This type of control strategy is known as a "feedback linearization" method. It transforms the nonlinear robot dynamics into a linear system in polar coordinates and then applies a linear control law. The complete code with animation can be referred in APPENDIX B.

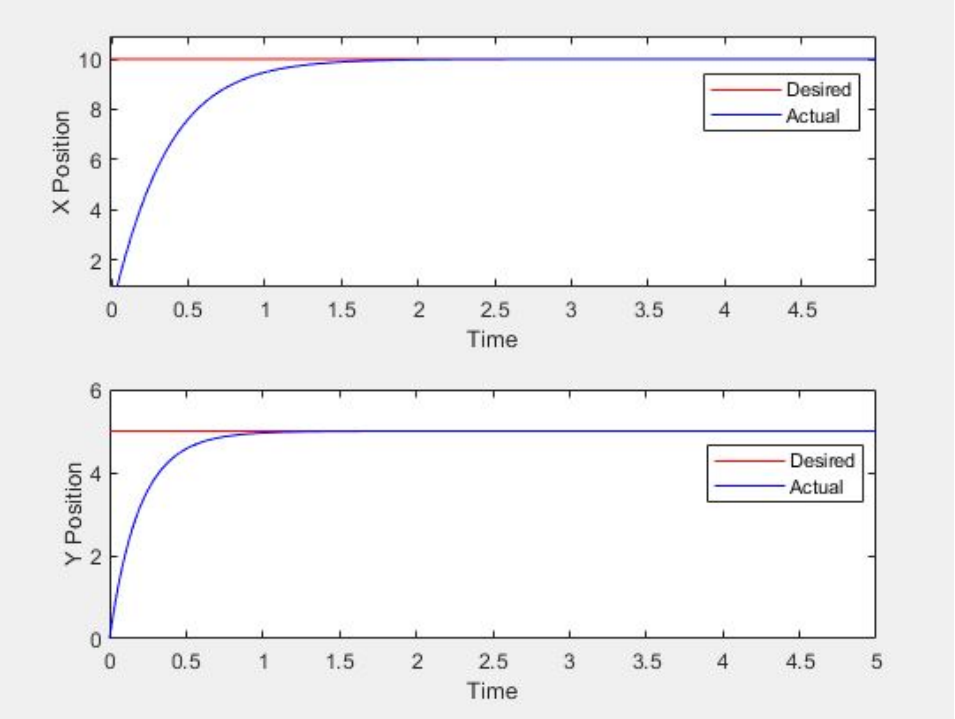

Figure 6.7: x, y Position Plot from t_0 to t_5

6.3 Waypoint Tracking Control of Dandebot using MATLAB

Building upon the kinematic models, this section presents a practical implementation of a PD-based waypoint tracking controller in MATLAB. The goal is to validate whether Dandebot can reach a specified target using computed wheel speeds. The simulation leverages the robot's physical dimensions and actuator limits, with visual plots illustrating the robot's trajectory and controller performance.

To validate the high-level control strategy of Dandebot before implementing hardware-level control, we designed and tested a MATLAB-based waypoint tracking controller. The purpose of this simulation was to confirm that the robot could reach a desired goal position accurately using only wheel velocity commands.

6.3.1 Kinematic Model and Control Architecture

Dandebot uses a differential drive system, modeled as a unicycle. The control inputs are the linear velocity v and angular velocity ω, which are mapped to left and right wheel angular velocities ω_l and ω_r. For the Dandebot prototype, the following physical parameters were used:

- Wheel radius: $r = 0.073$ m
- Distance between wheels (robot width): $d = 0.65$ m

The velocity transformation is defined by:

$$M = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ \frac{r}{d} & -\frac{r}{d} \end{bmatrix}, \quad \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix} = M^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{6.4}$$

6.3.2 Waypoint Controller Design

The goal was to drive Dandebot from its initial pose $q_0 = [x_0, y_0, \theta_0]^T$ to a target position (x_g, y_g). A Proportional-Derivative (PD) controller was used to compute the required velocities. Let e_p be the position error and e_θ be the heading error. The linear and angular velocities were calculated as:

$$v = k_{p1} e_p + k_{d1} \frac{de_p}{dt} \tag{6.5}$$

$$\omega = k_{p2} e_\theta + k_{d2} \frac{de_\theta}{dt} \tag{6.6}$$

The gains used were: $k_{p1} = 2$, $k_{d1} = 3$, $k_{p2} = 12$, $k_{d2} = 7$. Velocity saturation was enforced to ensure motor constraints:

$$|\omega_l|, |\omega_r| \leq 10 \text{ rad/s} \tag{6.7}$$

6.3.3 Simulation and Results

The controller was implemented in MATLAB and executed over discrete time steps using a Runge-Kutta integration method (‘ode45‘). Dandebot started at $(x_0, y_0) = (0, 0)$ and successfully reached the target waypoint $(1, 0.5)$.

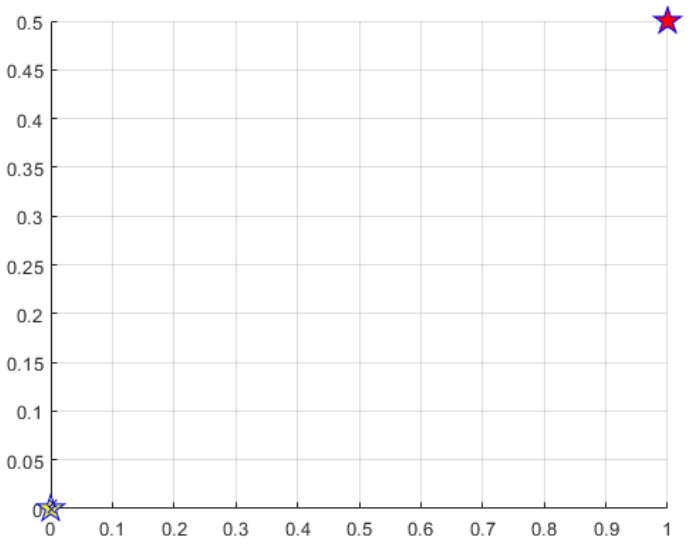

Figure 6.8: Dandebot's simulated initial position in waypoint tracking

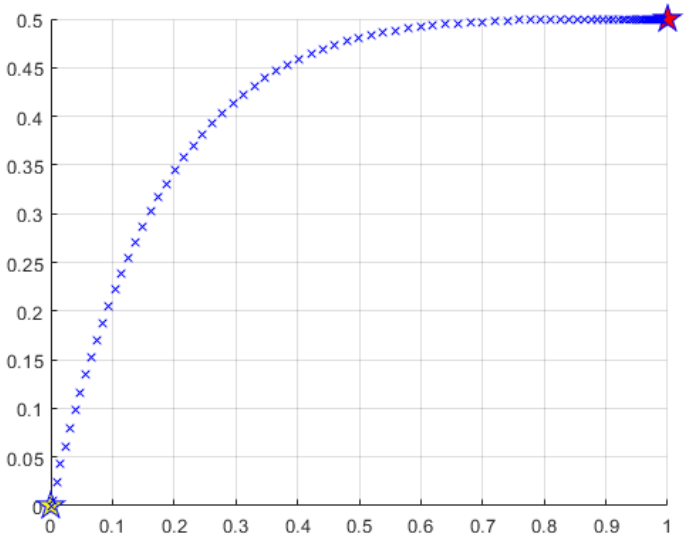

Figure 6.9: Dandebot's final position after successful waypoint tracking

Wheel velocities and heading were also tracked and plotted to observe system stability and controller performance:

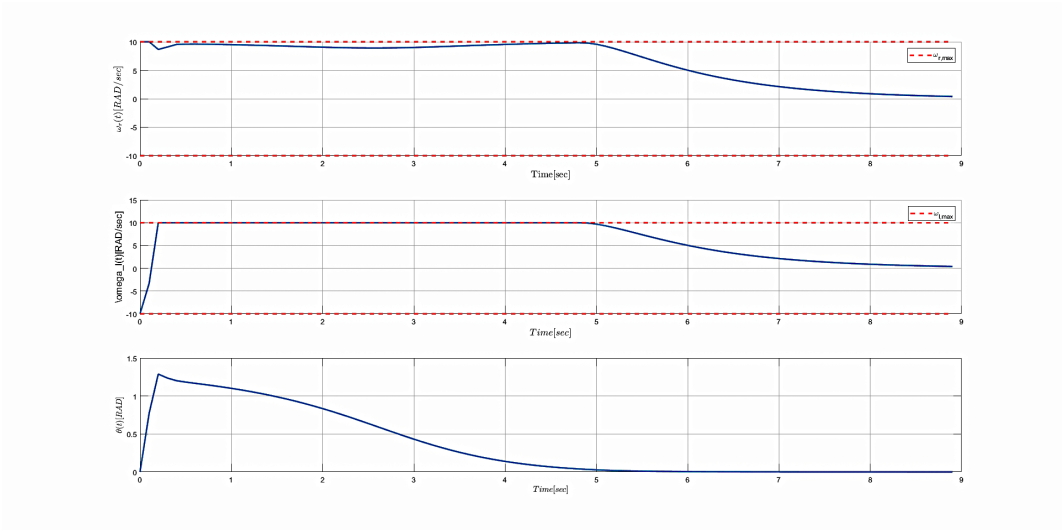

Figure 6.10: Angular velocities ω_r, ω_l and orientation θ over time

6.3.4 Discussion

The controller exhibited fast convergence and remained within actuator limits. The smooth evolution of θ and bounded ω_l, ω_r values indicate effective damping and tracking. This simulation validated the forward kinematic model and confirmed the suitability of PD waypoint control as a high-level guidance scheme for Dandebot.

6.4 Low-Level Motor Control with ODrive

This section explores the low-level motor control strategy implemented using ODrive controllers. It explains the cascaded PID architecture for position, velocity, and torque loops, detailing how each loop interacts to ensure accurate wheel control. The section also outlines the various control modes tested (velocity, torque, trajectory) and safety features like spinout detection and ramp limits.

6.4.1 Cascaded Style PID Controller

Structure: The ODrive motor controller uses a cascaded PID loop structure for each motor. This means there are three control loops nested inside each other: an outer position loop, a middle velocity loop, and an inner current (torque) loop. The output of each loop feeds into the next: the position controller sets

a velocity target, the velocity controller sets a current (torque) target, and the current controller commands the motor driver to produce the needed voltage. Depending on the chosen mode, some loops are bypassed.

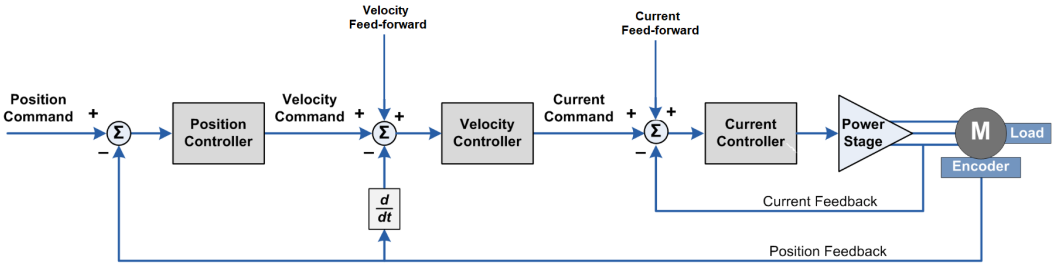

Figure 6.11: Cascaded position and velocity loops

For example, in position control mode, all three loops are active; in velocity control mode, the position loop is skipped and the velocity loop directly sets the current; and in torque (current) control mode, only the innermost current loop is used [43]. This cascaded setup allowed us to manage the wheel motion at multiple levels of responsiveness. Each loop is tuned separately, ensuring stable position tracking, smooth speed control, and accurate torque output. The structure can be seen in Figure 6.11.

Position Controller:

The position controller in ODrive's cascade is a simple proportional controller (P-only). It calculates a position error as the difference between the target position and the current encoder position [43]. The output of this controller is a velocity command. In formula form: $vel\_cmd = pos\_error * pos\_gain + vel\_feedforward$. Essentially, the further the wheel is from the desired position, the larger the velocity it commands (proportional to the error). In our differential drive robot, the direct position control was not often used (since they continuously rotate), but this loop is vital if you want the wheel to go to a specific angle or if using ODrive for something like steering or a robotic arm joint. The P-only nature means it responds to position error immediately, and any steady-state error is corrected by the downstream loops (velocity/current). It was found that keeping this loop P-only made it simpler to tune when combined with the other loops.

Velocity Controller:

The velocity controller is the next loop in the cascade. It is a PI controller (proportional-integral) that tries to make the wheel's actual speed match the commanded speed [43]. It computes a velocity error ($vel\_error = vel\_cmd - vel\_feedback$) and uses a proportional gain on that error plus an integrator term to eliminate steady errors. The output of the velocity loop is a current command (since more current produces more torque to increase speed). In the robot, this velocity PI loop was crucial when the ODrive was run in velocity control mode. For example, if the wheel was commanded to spin at 1.0 turn per second and was lagging, the velocity loop would increase current (torque) until the wheel reached the target speed. The P and I gains were tuned so that the wheel speeds would track the desired velocities quickly but without oscillation. The integrator term helped overcome friction or slight slopes by adding extra torque if the wheel was consistently below the commanded speed.

Current Controller:

At the core is the current controller, another PI loop that regulates motor current (which is directly proportional to torque) [43]. It takes the current command from the velocity loop (or directly from the user in torque mode) and drives the motor's voltage to achieve that current. It calculates $current\_error = current\_cmd - current\_feedback$ and adjusts the voltage via a proportional and an integral term [43]. Essentially, this loop deals with the electrical dynamics of the motor, ensuring the actual motor torque matches what is needed. ODrive automatically sets the gains for this loop based on the motor's characteristics (bandwidth configuration), so the default values provided by ODrive were primarily relied upon. In practice, this current loop reacted the fastest among the three – it can respond within milliseconds to load changes, giving more or less voltage to keep the motor torque on target. For the differential drive, having a tight current control loop meant that when the robot encountered a bump or heavier load on one wheel, the ODrive would immediately boost current to maintain wheel speed, providing smooth driving and torque balance.

6.4.2 Control Modes

Filtered Position Control:

If wheel positions were commanded directly in real-time, the motion could become jerky due to discrete updates. Filtered Position Control mode in ODrive addresses this by smoothing the input commands using a second-order filter [43]. Normally, ODrive holds each new setpoint until the next one arrives (a zero-order hold), which can cause a "staircase" effect in the commanded trajectory. With the filter enabled, those sharp steps are smoothed out. An input filter bandwidth was set roughly equal to the command update rate (e.g., $\tilde{2}0$ Hz) so that the filter would round off the edges of each step. This meant that when the high-level controller issued a sudden change in desired wheel position or a new small increment, the ODrive would interpolate between the old and new positions smoothly. The result was much less jolty wheel movement – the robot accelerated and decelerated more gently without sudden jerks. A feed-forward velocity term could also be added to reduce lag introduced by the filter. In practice, this mode proved helpful when experiments were conducted by sending position setpoints to the wheels (for instance, when treating wheel rotation like a small increment move); it improved ride smoothness by avoiding instant jumps.

Trajectory Control:

Trajectory Control mode uses ODrive's built-in trapezoidal motion planner. Instead of commanding an immediate move to a position, a target position is given, and ODrive itself generates a smooth velocity profile to reach it. The motion consists of a controlled acceleration phase, a constant velocity cruise, and a controlled deceleration phase, adhering to limits set for maximum speed and acceleration. Parameters such as vel_limit, $accel\_limit$, and $decel\_limit$ (in $turns/sec$ and $turns/sec^2$) were configured to define how fast the wheel should rotate and how sharply it can speed up or slow down. When this mode was used to, for instance, spin the robot's wheels a certain number of turns, the wheels would ramp up speed gradually, then coast, then slow down precisely to the stop – rather than lurching to full speed and slamming to a halt. This allowed the use of more aggressive PID gains for responsiveness (since the motion planner maintains smooth transitions) without making the ride rough. For example, feedback loops could be tuned to correct disturbances quickly, while the trajectory control ensured that the overall commanded motion remained gentle on the mechanics. This mode was mainly utilized for testing fixed rotations of the wheels; during autonomous driving, velocity control (with

the high-level planner handling the motion profile) was more commonly used. Nevertheless, it is beneficial to know that ODrive is capable of handling smooth point-to-point moves internally.

Circular Position Control:

For wheels that can spin indefinitely, Circular Position Control mode was very useful. In normal position mode, the encoder count (and *input_pos*) would continuously increase or decrease without bound as the wheel spins, which could eventually lead to numerical issues or loss of precision. In circular mode, ODrive is instructed to treat the position setpoints as wrapping around after one turn (or a defined rotation range). The $circular_setpoints$ parameter was enabled so that the controller interprets position commands modulo one revolution. For example, if the wheel's *input_pos* transitions from 0.9 to 1.1 turns, ODrive will wrap 1.1 around to 0.1 (one full turn rolled over) automatically. The controller then drives the motor to track that position within a single turn range. In the differential drive robot, this allowed a wheel to be commanded to an angle within [0, 360°) while continuously incrementing the target as the wheel rotated, without the internal target value growing endlessly. The encoder provides feedback in a wrapped sense (*pos_circular* instead of absolute count), so the controller always computes a small error within one turn. One caveat observed was that if a very large sudden step was issued (e.g., a command jump of more than one full rotation at once), the motor might take the shorter path around the circle, which could be in the opposite direction. However, for incremental motion, this behavior did not present an issue. Overall, circular position mode maintained stable wheel control during continuous rotation, preventing overflow and ensuring consistent precision over long runs.

Velocity Control:

This was the mode most frequently used for driving the robot. In Velocity Control mode, ODrive is configured to accept a target speed (turns per second) for each wheel [43]. The cascaded loops then act to maintain that speed: the velocity PI controller outputs the necessary torque to reach and hold the commanded velocity. Using this mode was straightforward – after setting $control\_mode = VELOCITY\_CONTROL$, the *input_vel* parameter was continuously updated with the desired wheel speed. For example, to drive the robot forward, both wheel velocities were set to a positive value; to rotate

in place, one wheel received a positive velocity while the other was assigned an equal negative velocity. ODrive then handled the rest, adjusting motor current to maintain the speeds even as the load changed (e.g., transitioning from carpet to tile). Due to the cascaded control structure, velocity mode implicitly employs the inner current loop to apply exactly as much torque as needed. This mode was found to be convenient for ROS integration: the ROS navigation stack outputs linear and angular velocity commands, which were converted into left/right wheel speeds and passed to ODrive. The robot's motion remained smooth and closely followed the commanded speeds due to the tight velocity regulation.

Ramped Velocity Control:

Ramped Velocity Control is a gentler variant of velocity mode. In this mode, instead of instantaneously jumping to a new speed command, ODrive gradually ramps the motor velocity at a specified acceleration rate. This was activated by maintaining the control mode in velocity control while changing the *input_mode* to VEL_RAMP and specifying a *vel_ramp_rate* (in $turn/s^2$) [43]. In the implementation, this mode was considered for limiting how quickly the robot could accelerate or decelerate, serving as an additional safety and smoothness feature. For example, if the robot was stationary and received a sudden command to move at full speed, the ramped mode would internally distribute that speed change over a short duration rather than executing an immediate jump. This helped prevent wheel slip and reduced mechanical stress. A ramp rate was selected based on the maximum acceleration the robot's wheels could handle without losing traction. During operation, if an emergency stop or sharp acceleration was required, it was handled directly by the high-level controller; however, the ramp mode functioned as a safeguard to ensure that no command caused an unrealistic transition. It effectively served as an onboard acceleration limiter within the motor controller. With ramping enabled, newly issued velocity commands resulted in smooth transitions of wheel speed, contributing to more graceful starts and stops in the robot's motion.

Torque Control:

In Torque Control mode, motor torque is directly commanded (which ODrive represents via motor current). This mode bypasses the position and velocity loops; ODrive utilizes only the current controller to achieve the commanded

torque [43]. The motor's torque constant (which converts Nm to current) had to be set first so that ODrive could correctly interpret the input value. For example, after configuring the torque constant for the motors, a command of $input\_torque = 0.1$ would instruct ODrive to apply approximately 0.1 Nm of torque to that wheel. This mode was not used during normal navigation, as velocity control is more directly applicable for maintaining speeds. However, torque mode proved valuable for low-level testing and for understanding the drive system. For instance, equal torque could be commanded to both wheels to observe the robot's response, or torque could be deliberately limited to test slip conditions. By default, ODrive imposes a velocity limit even in torque mode—if the wheel spins too fast under a torque command, the system reduces torque to prevent uncontrolled acceleration. This acts as a safety feature to avoid runaway scenarios when torque is commanded with no load. This limiter was left enabled, meaning that at stall or low speed, full torque was available, but if the wheel began spinning freely, ODrive would reduce torque accordingly. In summary, torque control provided direct access to force output for experimental purposes, while for actual driving, desired forces were typically converted into velocity targets using higher-level control.

6.4.3 Position Reference Mode

Startup-Relative Reference Frame:

Position reference frames define what "position = 0" means for the motor controller. By default, ODrive uses a Startup-Relative frame, which essentially zeroes the encoder position at the moment the system boots up or the motor is engaged [43]. In this mode, whenever a position is commanded (e.g., 10.0 turns), it is interpreted relative to the wheel's position at startup. For instance, if the wheel has not moved since startup, commanding 2.0 turns would result in a 2-turn forward rotation from that initial angle. This default setting was found to be convenient for a differential drive system, where relative movement (i.e., how far the wheels have turned from their starting position) is typically the main concern for odometry. At every startup, the wheels' current angles became the zero reference, and subsequent position readings ($pos\_estimate$) increased or decreased accordingly. This configuration is particularly suitable for "move X amount" commands. However, one consideration is that since the controller lacks an absolute reference, a power cycle will reset the wheel's

zero position to its current angle. In practical applications, this behavior was acceptable, as the high-level system required only incremental wheel motions, and the overall pose was recalibrated through other means, such as odometry resets or localization systems.

Homed Reference Frame:

The Homed frame is used when limit switches or end-stops are available to serve as a known reference. In systems such as linear axes or robotic arms, the motor is driven until a switch is triggered, and that position is defined as the "home" (typically set to zero). ODrive supports this by allowing the axis to perform a homing routine on startup: the motor moves until it activates an endstop sensor, then sets that position as a defined value (often 0.0) [43]. Once homed, all position commands are interpreted as absolute distances from the home position. In the case of the differential drive robot, the wheels were able to spin freely without a physical stop, so homing was not used for the drive wheels, as there was no built-in endstop for reference. However, if a steering mechanism or actuator with physical limits had been present on the robot, the Homed frame could have been employed to ensure that the system knew its exact physical position at each startup. Using this mode required setting $absolute\_setpoints = True$ to inform ODrive that an absolute reference would be used, along with configuring the appropriate endstop pins. Once enabled, ODrive would refuse to operate the motor in closed-loop mode until the homing process was completed, as a safety measure. While not directly implemented in the wheeled drive system, understanding this mode proved useful as it provides a method for achieving a consistent reference across restarts, provided the mechanism supports homing actions.

Custom User Reference Frame:

The Custom User frame allows the reference position to be set by an external source or procedure. Essentially, the user (i.e., the developer) can write a value to the motor's *pos_estimate* at startup to define what counts as zero or any other reference point. This is useful when an independent method is available for determining the motor's position. In a differential drive context, one possible implementation could involve using a vision system or a known marker on the wheels to set an exact angular reference upon robot startup. For example, if a wheel had a physical mark and a sensor capable of detecting a "calibration

position," the system could rotate the wheel to that mark and then manually set $pos\_estimate = 0$ to designate that orientation as zero. A custom reference for the wheels was not implemented, as it was not required for normal operation; however, the concept was tested by manually setting $pos\_estimate$ via ODrive tools to observe its effect. When this was done, $absolute\_setpoints = True$ was also set to ensure that ODrive waited for external input before accepting any position commands. This prevented accidental motion until the reference was established. In summary, the Custom frame is akin to instructing ODrive to "trust the externally provided position"—with the user supplying the ground truth position at startup. This method provides flexibility for integrating external calibration routines, if ever required.

Index-Based Reference Frame:

The motors' encoders used in the system have an index pulse—a special encoder signal that occurs once per revolution—and ODrive's Index-based frame leverages this to zero the position. When this mode is enabled, ODrive performs an index search on each startup: the motor rotates slowly until the encoder's index pin is detected [43]. Once the index is located, that position serves as a fixed reference for zero. This method was utilized on the differential drive wheels, as it provides a repeatable home position without requiring external sensors or mechanical end stops. Even though the wheels are capable of continuous rotation, the index pulse provides one known alignment per turn. Therefore, after completing the search, the controller recognizes that "at this particular wheel angle, the encoder index occurred, which is defined as position 0." As a result, if the robot is power-cycled and the wheels are manually moved, the index search during the next startup will recalibrate any drift, realigning the internal position count to the same physical angle as before. This improved odometry consistency across runs by preventing the accumulation of unknown offsets after restarts. The trade-off is a small delay at startup while the wheel performs the index search (a brief motion to detect the pulse), but this was considered acceptable in exchange for the precise reference. The index search routine was triggered on startup, either manually or set to occur automatically. Once the search was completed, ODrive allowed closed-loop control, and all position commands and references became relative to the index-calibrated zero.

6.4.4 Spinout Detection

"Spinout" refers to a loss of effective control, such as a wheel spinning in place or an encoder misreading, which can occur if the robot's wheel loses traction or the sensor slips. ODrive includes a Spinout Detection feature that continuously monitors whether the motor's electrical effort is resulting in the expected mechanical motion. This is achieved by comparing commanded mechanical power against the actual electrical power input. If the motor is delivering high current but the encoder feedback does not reflect the expected movement—indicating possible wheel slippage or encoder malfunction—ODrive flags a spinout condition. In such instances, it will fault and disable the motor for safety. In the differential drive robot, this served as a valuable safety measure. For example, if the robot became stuck against an obstacle and one wheel began slipping on the surface, or if an encoder became unplugged and failed to provide feedback, the system would detect the anomaly and cut power, triggering a $SPINOUT\_DETECTED$ error. Encoders were mounted securely to avoid any slippage on the shaft, as the documentation highlights encoder slippage as a common cause of false spinouts. Encoder offsets were also recalibrated whenever the motor was disassembled and reassembled, since misalignment could lead to erroneous spinout detection. During high-demand maneuvers—such as sudden acceleration on a low-traction surface—a spinout fault was encountered once when the wheel spun faster than the robot's forward motion (akin to a brief burnout). ODrive responded by stopping the motor, preventing uncontrolled wheel flailing. In scenarios where the system was known to be functioning correctly—such as with very low-resolution Hall effect encoders that occasionally triggered the fault during quick movements—ODrive allows the adjustment of sensitivity thresholds. However, the default settings were generally retained, favoring a conservative approach. This feature added robustness to the robot's drive system by automatically halting operation if the motors failed to behave as expected relative to sensor feedback, thereby avoiding potential damage or erratic motion.

6.5 High-Level Navigation Control in ROS 2

The final section of the control chapter focuses on the high-level navigation stack based on ROS 2's Navigation2 framework. The Dynamic Window Approach

(DWB) controller is used to evaluate and select optimal trajectories based on robot dynamics, obstacles, and goal alignment. This section also describes trajectory generation plugins and critic functions that guide Dandebot's decisions in real-time navigation tasks.

6.5.1 DWB Controller

For high-level motion planning and control, the robot utilizes ROS 2's Navigation2 stack. The DWB Controller (Dynamic Window Approach controller) was selected as the local trajectory controller. It is the default controller in Nav2 and serves as a ROS 2 implementation of the classic Dynamic Window Approach used in ROS 1, tuned specifically for differential drive robots [66]. The DWB algorithm operates by generating a set of possible short trajectories based on different combinations of wheel speeds and scoring each one to select the most optimal option. On each control cycle (approximately 20Hz in this implementation), DWB receives the robot's current pose and velocity along with the global path to follow. It then samples potential velocity commands combinations of linear and angular velocities within defined limits [67].

For each candidate velocity, the controller predicts the resulting trajectory over a short horizon (a few seconds), essentially simulating where the robot would end up if it continued at that speed. These simulated trajectories are then evaluated using a collection of cost functions, referred to as "critics," which assess criteria such as potential for collision, deviation from the global path, and progress toward the goal. Each trajectory is assigned a score, and the one with the lowest cost indicating the best adherence to operational constraints and objectives is selected. The DWB controller then outputs the corresponding wheel speeds as velocity commands to the robot.

In this system, the velocity commands produced by DWB were passed to the low-level ODrive velocity controller, thereby completing the loop from high-level planning to motor execution. The DWB controller was found to be effective in smoothly tracking the global path while dynamically avoiding obstacles. Additionally, it offered high tunability through its plugin-based architecture, which was configured to align with the robot's dynamic properties. In essence, DWB functioned as the "brain" determining the immediate motion strategy (e.g., turning slightly left, reducing speed), while the ODrive controller ensured

precise execution of that strategy at the hardware level.

6.5.2 Trajectory Generator Plugins

Standard Trajectory Generator:

DWB's first task is to generate candidate trajectories, and the StandardTrajectoryGenerator is one available plugin for this purpose. This generator is similar to the trajectory rollout approach from ROS's *base_local_planner* [67]. It samples a range of possible linear and angular velocities, then "rolls out" the robot's motion forward in time for each velocity pair. The sampling is not heavily constrained by the current velocity; instead, a dense set of combinations (within defined minimum and maximum limits) is explored, with a trajectory generated for each.

For a differential drive configuration, StandardTrajectoryGenerator considers various forward speeds (e.g., from 0 up to 0.26 m/s) and rotational velocities (e.g., from –1.0 to 1.0 rad/s), including combinations of forward and turning motions. Each combination is simulated over a short horizon (typically 1–2 seconds) to evaluate where the robot would end up if those speeds were sustained. The result is a grid of possible arcs or paths the robot could follow. This generator was initially used because of its straightforward design and comprehensive coverage of the velocity space.

One consideration is that the generator may produce trajectories that require sudden changes in acceleration, as it does not explicitly account for whether a velocity can be reached instantaneously it functions as an open-loop simulation. However, in practice, the DWB controller continues to enforce the robot's configured kinematic limits, ensuring that no velocity commands exceed the specified constraints. Although StandardTrajectoryGenerator does not inherently model acceleration between time steps, it proved sufficient when the velocity limits and controller frequency naturally limited abrupt changes. For more dynamic maneuvers, the limited acceleration option was also evaluated as a potential alternative.

Limited Accel Generator:

The LimitedAccelGenerator is the trajectory generator plugin that adheres more closely to the original spirit of the Dynamic Window Approach (DWA)

by incorporating the robot's acceleration limits when proposing trajectories [67]. This generator is similar to the original DWA implementation from ROS. Rather than sampling arbitrary velocities across the full range, it evaluates what velocities are achievable within the next time step, based on the robot's current velocity and defined acceleration/deceleration constraints. Conceptually, it operates by asking: given the current speed, how much can the robot speed up or slow down in the next 0.1 seconds? This defines a dynamic "window" of reachable velocities, within which the generator samples and projects candidate trajectories.

In this system, acceleration limits were configured, e.g., maximum linear acceleration of 2.5 m/s^2 and maximum angular acceleration of 3.2 rad/s^2, as defined in the parameter set. For a stationary robot, the first control cycle would consider only those velocities attainable within a 0.1-second acceleration to 2.5 m/s^2, resulting in a modest initial speed range. As the robot increases speed over time, the reachable velocity window expands, enabling exploration of higher velocities in later cycles. This results in candidate trajectories that are physically feasible and do not rely on instantaneous velocity changes.

The LimitedAccelGenerator was preferred during tuning because it naturally produced smoother command sequences. It avoided abrupt transitions to full speed in a single step, allowing the local planner to generate a gradual acceleration profile. This behavior proved beneficial in scenarios with tight navigation constraints, where abrupt changes could lead to oscillatory motion. By constraining the velocity search space to a realistic and narrower set of options, the planner exhibited improved stability in its decisions. In summary, the LimitedAccelGenerator ensured that DWB only evaluated feasible and safe motion commands for the differential drive system, resulting in more reliable and stable navigation particularly valuable when aggressive acceleration limits were imposed for safety.

6.5.3 Critic Plugins

Base Obstacle Critic:

One of the most critical concerns for the robot is avoiding collisions. The BaseObstacleCritic is a plugin that scores each trajectory based on its interaction with the costmap, which represents the surrounding obstacles. Specifically,

it analyzes the predicted path of the robot and examines the costmap cells that the trajectory would traverse. Trajectories passing through free space receive a low cost (indicating a good path), whereas those passing near or into obstacles are assigned higher costs. Trajectories that directly intersect with occupied space are deemed invalid. An inflated costmap where obstacle regions are expanded based on the robot's radius is used, as recommended in documentation [68], so that even proximity to obstacles contributes to the trajectory's score.

The BaseObstacleCritic computes the sum of cost values for all the cells covered by a given trajectory. For example, a path that closely grazes a wall and passes through high-cost inflated cells will be rated poorly compared to a trajectory that remains in clear, low-cost areas. In the differential drive implementation, this critic enabled the local planner to effectively "sense" the presence of obstacles. Any trajectory that led toward a wall or a person was heavily penalized and typically excluded from selection.

This behavior was clearly observed during testing in confined spaces, such as narrow corridors paths that brought the robot too close to the sides were associated with significantly higher costs, prompting the DWB controller to favor slightly slower or more centrally aligned alternatives. The influence of this critic could be adjusted by tuning its weight (scale), thereby modifying how strongly the robot favored open space over shorter or riskier paths. Ultimately, the BaseObstacleCritic played a key role in ensuring safety by embedding obstacle avoidance directly into the trajectory evaluation process.

Obstacle Footprint Critic:

While the BaseObstacleCritic uses costmap values to assign costs to trajectories, the ObstacleFootprintCritic performs a more binary check for collisions. This critic ensures that at no point along a candidate trajectory does the robot's footprint intersect with any obstacle. It simulates the robot moving along the proposed path and evaluates the robot's physical shape (footprint) against static obstacles. If any overlap is detected indicating a potential collision—the trajectory is either invalidated or assigned a very large cost.

In this system, the ObstacleFootprintCritic functioned as a fail-safe. Even if a trajectory passed close to an obstacle and the costmap inflation was insufficient

to flag it as risky, the footprint-based check would still detect the actual risk of collision. The robot's footprint, configured as a rectangle appropriate for a differential drive model, was defined in Nav2 so the critic could accurately perform its checks against environmental geometry.

During testing, it was observed that if the inflation radius was accidentally configured too low, the BaseObstacleCritic might allow certain trajectories with slight overlaps. However, the ObstacleFootprintCritic consistently rejected any trajectory that would cause contact with an obstacle. This ensured that selected trajectories were not just low-cost but entirely collision-free. As a result, the local planner could be trusted not to knowingly drive the robot into an obstacle. When used in conjunction, the BaseObstacleCritic maintained a safety buffer around obstacles, while the ObstacleFootprintCritic enforced absolute collision avoidance, together enhancing the reliability and safety of the navigation system.

6.5.4 Goal Align and Goal Dist Critics

These two critics work together to guide the robot toward its final goal in an efficient and properly oriented manner.

GoalDistCritic scores trajectories based on how close the robot would get to the goal if it followed a given path [68]. Specifically, it evaluates the endpoint of each trajectory relative to the goal position. The closer the trajectory endpoint is to the goal, the lower (and therefore better) the assigned cost. This encourages continuous progress toward the target. When the robot is presented with multiple potential paths, the trajectory that results in a shorter remaining distance to the goal is rated more favorably by this critic. This prevents behaviors where the robot might take detours that do not contribute to goal convergence. Essentially, the local planner regularly assesses whether a proposed motion brings the robot closer to the goal, and GoalDistCritic quantifies that evaluation. The weight of this critic was configured such that the robot prioritized reducing the distance to the goal, except when obstacle avoidance temporarily necessitated an alternate route.

Together, the GoalDistCritic and GoalAlignCritic ensure that the robot not only advances toward the goal position at every step, but also begins orienting itself appropriately for a well-aligned arrival. These critics collectively provide

the local planner with a sense of moving "efficiently and gracefully" toward the objective.

6.5.5 Path Align and Path Dist Critics

These critics keep the robot adhering to the global path planned by the higher-level planner.

PathDistCritic assigns a cost based on how far a candidate trajectory strays from the global path. The global planner generates an optimal or safe path represented as a series of waypoints. PathDistCritic evaluates, for each simulated trajectory, the distance between the robot and the nearest point on the global path often considering the endpoint of the trajectory in relation to the path. If a trajectory deviates significantly from the intended route, its cost increases. In this system, the critic discouraged lateral deviations away from the intended navigation corridor. For instance, if the global path followed a straight line down a hallway and one trajectory diverged 0.5 meters to the left, it would receive a higher cost than another that remained within a few centimeters of the line. By tuning this critic, the robot was made to "hug" the path more closely, a useful behavior in tight environments where the global path is carefully selected to navigate between obstacles.

PathAlignCritic evaluates how well the robot's heading along a trajectory aligns with the orientation of the global path. It functions similarly to GoalAlignCritic, but focuses on the direction of travel along the path rather than alignment with the final goal. The critic examines a forward point on the trajectory and compares the robot's heading at that point to the orientation of the global path at the corresponding location. When the trajectory maintains alignment with the path direction, the cost remains low; if the trajectory diverges in orientation such as pointing away or forming a sharp angle to the path the cost increases. This promotes movement that follows the path's direction and discourages sharp or misaligned maneuvers. In practical scenarios, such as curved paths, this critic was especially helpful. Without it, the robot might attempt to turn while partially facing outward, thereby deviating from the curve. PathAlignCritic encouraged adherence to the curve's tangent, resulting in smoother and more natural motion.

Together, PathDistCritic and PathAlignCritic helped the differential drive

robot remain aligned with the global plan. These critics acted as a path fidelity mechanism, penalizing significant lateral deviations and heading misalignments. As a result, the robot's local motion decisions remained anchored to the intended route, except in situations where deviation was necessary, such as during obstacle avoidance at which point obstacle-related critics would temporarily take precedence.

Rotate to Goal Critic:

The RotateToGoalCritic handles the special behavior required when the robot is very close to the goal and needs to adjust its orientation accurately. This critic is designed to permit in-place rotation only when appropriate. During most of the navigation process, in-place spinning is undesirable; the robot is expected to make forward progress toward the goal. However, upon reaching the vicinity of the goal position, it often becomes necessary to perform a final rotation to match the specified goal orientation.

The RotateToGoalCritic enforces this behavior by assigning a high cost to trajectories involving in place rotation unless the robot is within a defined proximity to the goal. When the robot is far from the goal, any trajectory dominated by rotational motion particularly those involving spinning in place is penalized. This is because the critic "only allows the robot to rotate to the goal orientation when sufficiently close to the goal" [67]. Conversely, when the robot is near the goal, the conditions for this critic are satisfied, and pure rotational trajectories are permitted or even encouraged to align the robot's heading precisely with the goal orientation.

In practical implementation, this behavior was clearly observable. During normal motion, the robot rarely performed turns in place; instead, it executed gradual forward motion combined with smooth turning. Upon reaching the goal position, once the positional error became minimal, the local planner issued a spin-in-place command to achieve exact alignment with the goal orientation. This resulted in a behavior where the robot arrived at the target location and then performed a clean, stationary rotation to face the correct final direction.

Without the RotateToGoalCritic, the robot might have attempted the final orientation adjustment prematurely or skipped it altogether, resulting in inefficient movement or overshoot. By tuning this critic, the final orientation

adjustment was handled as a distinct, precise action, improving the clarity and efficiency of the robot's goal-reaching behavior.

Oscillation and Twirling Critics:

These two critics are designed to prevent undesirable oscillatory or excessive spinning behaviors in the robot's motion.

OscillationCritic addresses situations where the robot risks becoming stuck by repeatedly reversing direction without making meaningful progress. It monitors recent velocity commands and penalizes sudden reversals, such as switching from left to right or from forward to backward. This critic was effective in eliminating jittering behavior when the robot navigated through tight spaces. By increasing the cost of such oscillatory motions, the robot was encouraged to commit to a definitive direction rather than hesitate.

TwirlingCritic is intended to prevent the robot from performing excessive in-place rotations while moving toward a goal. Although differential drive robots do not typically spin in place as easily as omnidirectional platforms, this critic still proved beneficial by discouraging unnecessary turning. It applied a penalty to any excessive angular motion that was not required, thereby promoting more stable and direct orientation during travel.

6.6 Summary

This chapter presents the complete control strategy for Dandebot, an autonomous differential-drive robot. It begins with forward and inverse kinematics to model and simulate the robot's motion in MATLAB, followed by waypoint tracking using a PD controller. Low-level wheel control is handled by ODrive using cascaded PID loops across multiple control modes. At the high level, ROS 2's Dynamic Window Approach (DWB) controller enables real-time trajectory planning and obstacle avoidance using critic-based scoring. Together, these layers allow Dandebot to navigate smoothly and accurately toward mowing and weeding targets.

Chapter 7

Experimental Test and Validation

7.1 Introduction

This custom-built lawn care robot is designed on a ROS2 Iron platform and equipped with a robust set of hardware as shown in Figure 7.1. It features a custom differential-drive chassis for movement, a 2D LIDAR for scanning the surroundings, a camera for vision, and an IMU for orientation and stability.

Figure 7.1: Dandebot

Attached to the centre of the chassis is a mechanical weed extraction tool inspired by the Fiskars stand-up weed puller (a device with claws that remove weeds by the root). The following sections describe six advanced functionalities implemented on this robot, explaining how each works and why it is important.

7.2 Remote Teleoperation

Remote teleoperation allows an operator to manually control the robot from a distance, which is invaluable during setup, testing, and when handling edge-case scenarios. The robot leverages ROS2 Iron's teleoperation tools to achieve this. For example, a teleop ROS2 package (such as $teleop_twist_keyboard$ or a gamepad teleop node) reads user input from a console controller as shown in Figure 7.2 and publishes velocity commands to the robot's drive system.

Figure 7.2: Xbox Joystick Controller

At the same time, the robot's onboard camera stream is sent back to the operator to provide visual feedback. In practice, the operator might run a

control interface on a laptop or tablet that shows the live video feed alongside control widgets. This could be done through RQt (ROS's Qt-based UI tools) or a web interface using WebRTC for low-latency streaming. The goal is to let the person see what the robot sees and drive it in real time. This real-time visual feedback and control are crucial for safety and precision in teleoperation.

During initial setup, teleoperation is used to drive the robot around the yard to test components and even to help generate a map of the area. It is much like having a remote-controlled mode: the developer or user can steer the robot to explore the environment, verify sensor data, and calibrate systems.

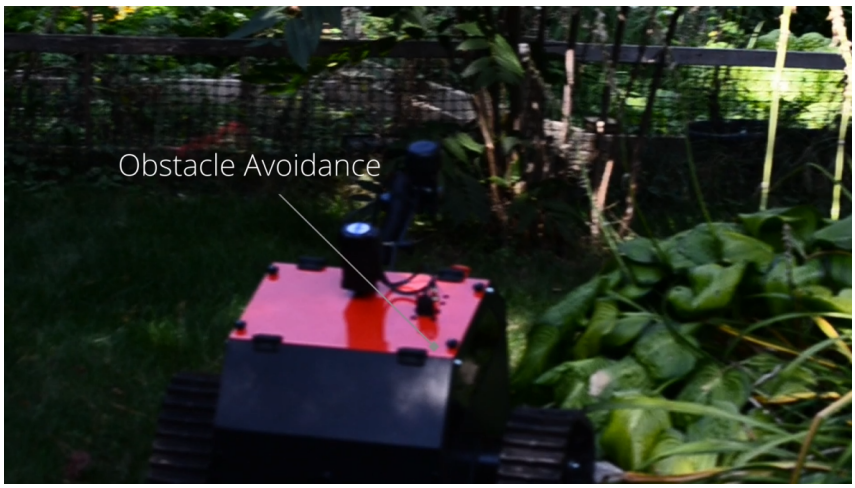

Figure 7.3: Unnoticed entry into a flower bed

Teleop control is also a safety net for edge cases. If the robot ever gets into trouble – for instance, stuck in a tight corner, encountering an unexpected obstacle as shown in Figure 7.3, or experiencing a software glitch – a human operator can take over via remote control to navigate the situation. This human-in-the-loop intervention ensures that the robot can be recovered from anomalies without having to physically pick it up. In summary, remote teleoperation combines a ROS2 teleop control node (for sending drive commands) with remote video feedback, giving the operator eyes on the robot. It's an essential feature that complements autonomous operation by allowing manual override and supervision whenever needed.

7.3 Intelligent Mapping System

For the robot to navigate autonomously, it needs to understand its environment. This lawn care robot uses an intelligent mapping system based on SLAM (Simultaneous Localization and Mapping) to build and continuously update a map of the lawn. In ROS2 Iron, there are SLAM solutions like SLAM Toolbox (optimized for 2D LIDAR-based mapping) and RTAB-Map (which can fuse LIDAR and camera data for 2D/3D mapping). The robot takes advantage of its sensor suite – LIDAR, camera, and IMU – in combination to perform SLAM. Here's how it works in practice:

7.3.1 Sensor Fusion for Localization

The robot's wheel encoders provide odometry data, and the IMU provides information about its orientation and acceleration. These are fused (often using a package like *robot_localization* with an Extended Kalman Filter) to give a more accurate estimate of the robot's pose (position and heading) over time. By fusing continuous data from encoders and IMU, the system reduces drift and knows how far the robot has moved and rotated. This fused odometry is the backbone for both mapping and navigation, ensuring the robot's movements are tracked even on bumpy or slippery terrain.

The implementation of EKF was done through studying and analyzing mathematical concept behind it. The EKF maintains a state vector x that typically includes:

$x = [x\_position, y\_position, theta,$ linear_velocity, angular_velocity, $\dots]$

It also maintains a covariance matrix P representing uncertainty in the estimate.

Prediction Step (based on motion model)

The motion model uses velocity information from encoders to predict the next state of the robot. For a differential drive system:

$$x_{k+1} = x_k + v * \Delta t * cos(\theta) \tag{7.1}$$

$$y_{k+1} = y_k + v * \Delta t * sin(\theta) \tag{7.2}$$

$$\theta_{k+1} = \theta_k + \omega * \Delta t \tag{7.3}$$

Where:

v is the linear velocity (from wheel encoders),

ω is the angular velocity,

θ is the current heading,

Δt is the time interval between updates.

This predicted state is based on **dead reckoning** and is subject to error due to slippage and sensor noise.

Update Step (based on sensor measurements)

The IMU provides independent measurements of angular velocity and linear acceleration. These are used to correct the predicted pose from the encoder-based estimate. The EKF compares the expected measurement from the predicted state with the actual IMU reading:

$z_k = h(x_k) + noise$

Here, z_k is the actual measurement (e.g., IMU yaw rate), $h(x_k)$ is the predicted measurement from the current state, and noise models sensor uncertainty.

Using the Kalman Gain K, the filter corrects the predicted state:

$x_k = x_k + K * (z_k - h(x_k))$

Where K is computed based on the uncertainties in both prediction and measurement (P, and the measurement noise matrix R).

Covariance Update

After each update, the uncertainty P is updated to reflect how confident the system is in its new estimate:

$P_k = (I - K * H) * P_k$

where H is the Jacobian of the measurement model.

The practical setup configuration looks like the below YAML example:

```
1 ekf_filter_node:
2   ros_parameters:
```

```
    frequency: 30.0
    two_d_mode: true
    sensor_timeout: 0.1
    odom0: odom
    odom0_config: [true, true, false,
                   false, false, true,
                   false, false, false,
                   false, false, false,
                   false, false, false]
    imu0: imu/data
    imu0_config: [false, false, false,
                  true, true, true,
                  false, false, false,
                  false, false, false,
                  false, false, false]
    publish_tf: true
    map_frame: map
    odom_frame: odom
    base_link_frame: base_link
```

7.3.2 LIDAR SLAM for Mapping

SLAM, or Simultaneous Localization and Mapping, is a method that allows the robot to build a map of an unknown environment while also keeping track of its own position on that map. This is essential for autonomous operation, especially in outdoor environments like lawns where GPS may be unavailable or unreliable. In this project, SLAM Toolbox is used as the main package for SLAM in ROS2 Iron, and it processes the data from the robot's 2D LIDAR and IMU to create a live map of the yard. Some of the maps used in the project are the ones from a backyard and the university's corridor as shown in Figure 7.4 and 7.5.

Figure 7.4: Backyard Map

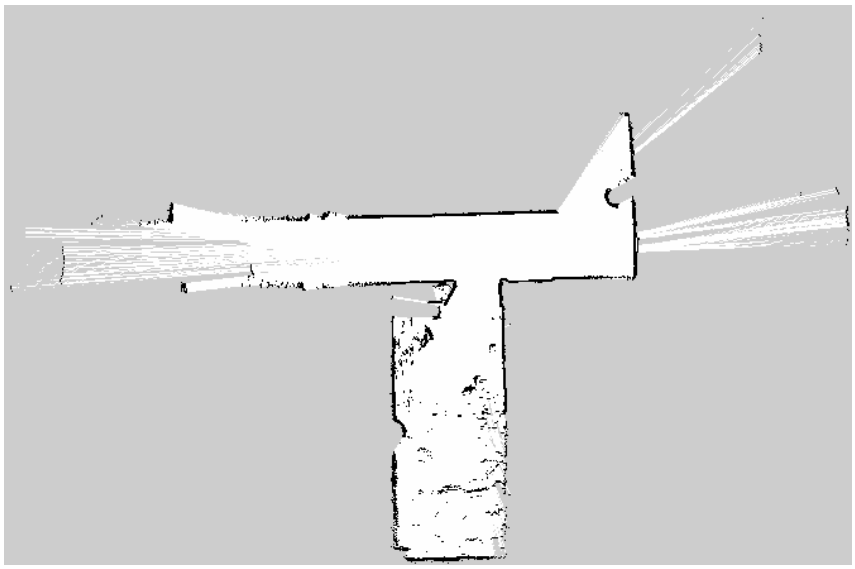

Figure 7.5: Lab Corridor Map

LIDAR-Based Scan Matching

The LIDAR continuously emits laser pulses and measures how long it takes for the light to bounce back after hitting an object. This provides a set of distance measurements in all directions (usually 360°). Each time the LIDAR completes a full rotation, it produces a scan — a 2D array of ranges at different angles.

Each LIDAR scan is treated like a "snapshot" of the surroundings. When the robot moves forward, a new scan is generated. The SLAM algorithm compares the new scan to the previous one using scan matching to estimate how the robot moved between the two scans. This is done using algorithms like:

- **ICP (Iterative Closest Point):** aligns points in the current scan with the previous scan.
- **Correlative Scan Matching:** tries different poses and scores how well the scans overlap.

These pose estimates are then used to update the robot's trajectory on the map.

Pose Estimation with Sensor Fusion

As the robot moves, it also uses data from the wheel encoders and the IMU (fused using the Extended Kalman Filter, as explained earlier). This gives an initial guess of the robot's motion. The scan matching then refines this guess by comparing actual scan data to the expected scan at that pose.

SLAM Toolbox uses this fused pose information to insert new scans into the map, with minimal drift. Over time, the robot builds a pose graph, which is a chain of poses connected by motion estimates.

Map Representation – Occupancy Grid:

The map created by SLAM Toolbox is a 2D occupancy grid. This grid divides the space into small squares (cells), each of which can be:

- **Occupied (obstacle detected – like a tree, wall, or garden border)**
- **Free (no obstacle – the robot can move there)**
- **Unknown (area not yet scanned)**

This grid forms a bird's-eye view of the yard. It looks like a floorplan, where obstacles like fences, flower beds, and lawn edges are drawn based on LIDAR data.

As more scans are added, the map becomes denser and more accurate. The robot avoids mapping the same thing twice by referencing existing cells in the grid and only updating when confident.

Loop Closure and Map Correction:

Drift naturally occurs when a robot moves for a long time based on its own sensors. SLAM Toolbox handles this by doing loop closure — the process of detecting when the robot has returned to a previously visited location.

If loop closure is detected (for example, the robot finishes a full loop around the yard), the algorithm adjusts all the poses in the graph to correct accumulated error. This creates a more consistent and accurate map by aligning the newly scanned features with old ones. It's like connecting the ends of a drawn circle when they meet again and smoothing the whole shape.

Using the Map in ROS2

Once the map is built:

- It is published as a ROS2 topic (*/map*) in the form of an OccupancyGrid message.
- Navigation algorithms (like Nav2) subscribe to this map to plan safe paths.
- The robot can load this map in future sessions using slam_toolbox's serialization tools (*load_map*, *save_map*).

A simple example of launching SLAM Toolbox in ROS2 Iron:

```
ros2 launch slam_toolbox online_async_launch.py
use_sim_time:=false
```

7.3.3 Camera Integration

In addition to LIDAR, the onboard camera can contribute to the mapping process, especially if a visual SLAM approach is used. RTAB-Map, for instance, can use the camera to recognize visual features and build a richer map (including 3D point clouds or color information). In an outdoor lawn scenario, the camera might pick up distinct landmarks (like the house walls, flowerpots, or patio edges) which can help in localization. The camera's data can also be used to create a 2D occupancy grid if paired with depth estimation, but in this robot the primary mapping sensor is the LIDAR for reliability. Nonetheless,

the fusion of camera with SLAM can improve accuracy in certain cases – for example, if part of the yard has low LIDAR features (like a flat open lawn with few obstacles), visual cues can assist in positioning.

7.3.4 Real-Time Map Updates

The mapping system runs continually as the robot operates. This means the map is not static – the robot updates it on the fly. If a new obstacle appears in the yard (say someone leaves a wheelbarrow or a lawn chair out), the LIDAR will observe it and the SLAM algorithm will mark that area as occupied on the map. Modern SLAM tools like SLAM Toolbox support lifelong mapping, where the map can be maintained and even updated or refined over multiple sessions. In effect, the robot ends up with a "living" map of the lawn. Each time it runs, it can load an existing map and refine it further or adjust to changes. The outcome is an accurate, up-to-date representation of the environment.

Once this map is built, it serves as a foundation for navigation tasks. The ROS2 navigation stack (Nav2) uses the occupancy grid map to plan paths for coverage, point-to-point travel, obstacle avoidance, etc. In summary, the intelligent mapping system ensures the robot always knows where it is (localization) and what the environment looks like (mapping). By fusing LIDAR, IMU, and optionally camera data, the robot achieves robust SLAM, enabling it to autonomously traverse the lawn with knowledge of its surroundings.

7.4 Complete Coverage and Weed Detection

One of the primary jobs of the lawn robot is to cover the entire lawn area to perform tasks like mowing and scanning for weeds. This requires a complete coverage path planning system so that the robot systematically moves over all sections of the lawn without missing spots. Simultaneously, the robot runs a weed detection algorithm to spot and mark weeds as it goes. These two capabilities work hand-in-hand: the coverage planning drives the robot to patrol the whole lawn, and the detection system finds the weeds during that patrol. A planned path (green solid line) for the corridor map can be seen in Figure 7.6 and 7.7.

Figure 7.6: Coverage Plan Example

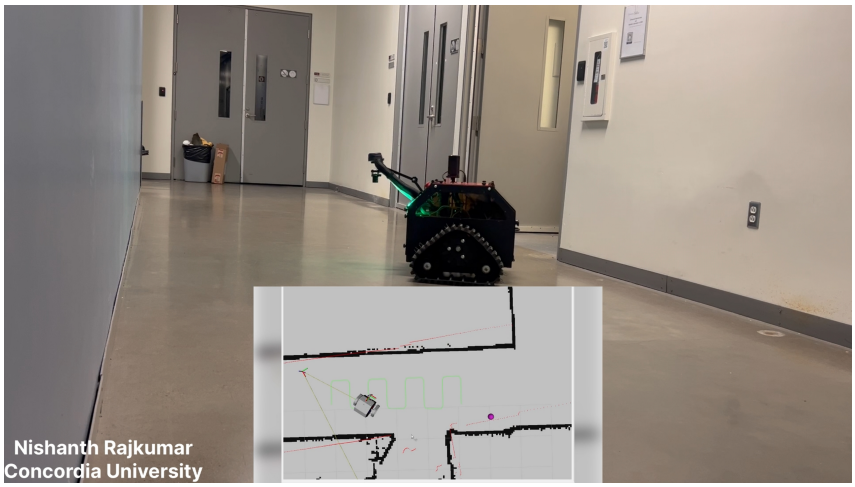

Figure 7.7: Coverage with Robot

7.4.1 Fields2Cover as Base Planner

To generate these systematic paths, the project integrated a planner built using the Fields2Cover library. Fields2Cover is an open-source library designed to generate efficient coverage patterns, mainly for agricultural robots. It takes a polygonal area (the lawn boundary) and creates a set of back-and-forth paths, known as boustrophedon patterns.

The planner overlays a virtual grid or lattice on top of the map of the lawn.

The planner then filters out areas that contain obstacles (like trees, furniture, or flower beds), so the paths avoid them. The remaining free space is divided into strips, and paths are planned along each strip, covering the full area while minimizing turning.

7.4.2 Designing a Planner for Dynamic Obstacle Handling:

While Fields2Cover provides a good static path, it does not handle dynamic obstacles, objects that might appear after the path is generated, like a person walking, a moving pet, or a chair that was not in the map before. To solve this, a coverage planner was built on top of Fields2Cover.

This custom-designed planner does two things:

- It generates the initial coverage waypoints using Fields2Cover.
- Then, during runtime, it checks for dynamic obstacles using real-time sensor input (from LIDAR and camera) and dynamically replans or bypasses that area when needed.

This was implemented by running the planner inside a modified behavior tree (BT) in the Nav2 stack. The default Nav2 behavior tree is designed to reach a single goal and stop. For this robot, the tree was restructured to:

- Accept a sequence of goals (waypoints from the coverage path).
- Navigate to each in order.
- Check for obstacles dynamically at each waypoint.
- If an obstacle is detected, skip, delay, or replan that waypoint.
- Return to the skipped area later when it's safe.

This new behavior tree introduces a node called something like CheckDynamicObstacle that uses LIDAR costmaps and real-time camera data to decide if the waypoint is currently blocked.

7.4.3 Runtime Obstacle Handling

The robot builds a local costmap around itself as it moves. This is continuously updated with LIDAR data. If something like a child or dog appears in the robot's path, the costmap will reflect this as an obstacle. When this happens:

- The NavigateToPose action in the BT is paused or rerouted.
- The robot can temporarily skip that area or replan around the obstacle.
- After the dynamic object leaves, the skipped waypoint is added back to the queue.

This ensures that the robot never gets stuck or hits anything and still achieves full coverage.

In summary, the Complete Coverage system guarantees that the robot physically passes over the entire lawn area, and the Detection system ensures that as it does so, it will spot any weeds in view. By the time the robot has finished covering the lawn, it will have a list of coordinates for all detected weeds (if any). At that point, it can switch modes to go remove them. The next section describes how the robot goes from simply noticing a weed to actually extracting it.

7.5 Point-to-Point Weed Extraction

Once a weed has been detected and its location noted (thanks to the coverage and detection systems), the robot engages its point-to-point (P2P) weed extraction functionality. This involves two phases: navigating to the weed's exact location, and then physically removing the weed with the mechanical tool. Both require precision and coordination within the robot's ROS2 system.

7.5.1 Targeted Navigation to the Weed

After a weed is identified as shown in Figure 7.8, the robot's autonomy software creates a goal at that weed's coordinates on the map. In ROS2 Nav2 terms, it's just like telling the robot "drive to this X, Y position". The navigation stack then plans a path from the robot's current position to the weed. This path planning uses the occupancy grid map to ensure the route is free of obstacles.

The Nav2 planner server will compute the shortest or most feasible path to the goal. For example, if a weed is on the other side of a flower bed, the planner will route around the bed. Under the hood, algorithms such as DWA Controller as explained in the Control chapter is employed by the global planner to find this path on the grid.

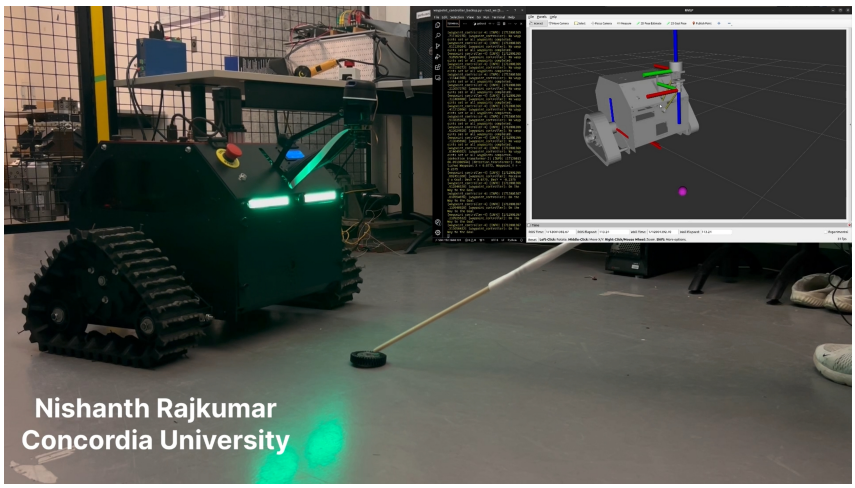

Figure 7.8: Target Locking After Identification

Once the global path is set, the local planner/controller takes over to drive the robot along that path. The robot will leave its coverage pattern and head directly to the weed, adjusting its wheels and heading as needed to stay on course. This transition from systematic coverage to goal-oriented navigation is smooth – the system essentially switches from "coverage mode" to "navigate-to-pose mode" using ROS2's behavior tree or state machine managing the mission. During this transit, the robot keeps the weed's position as the target and continually updates its course if needed (for instance, if an unforeseen obstacle appears, it will replan around it). The accuracy of the navigation is quite high; the robot can typically arrive within a few centimeters of the intended weed location.

7.5.2 Mechanical Weed Removal

On reaching the position of the weed with respect to the map, the robot triggers the weed extraction mechanism. The tool as shown in Figure 7.9 is inspired by the Fiskars 4-claw stand-up weeder, which uses serrated metal claws to grab a weed by the root when driven into the ground. In the robot's case,

a motor-driven mechanism pushes the claws into the soil around the weed. Typically, this would be a linear actuator or a geared motor that drives a plunger.

Figure 7.9: Weeding Toolhead

The claws penetrate the ground encircling the weed's root. Then the mechanism applies a clamping and lifting action – it might close the claws or rotate slightly to grip the root firmly, and then lift upwards, pulling the weed out of the ground. This mimics the human action of stepping on the weeder and pulling back to yank out the weed. The use of four opposed claws ensures the weed, especially something like a dandelion with a taproot, comes out entirely rather than breaking off. The ROS2 control for this tool is managed by a dedicated node or microcontroller. For instance, there could be a weed_extractor node that, upon command, publishes to an actuator topic to perform the sequence: drive claws down, clamp, lift up. Once the weed is uprooted, the robot can stow or discard it. A common design is to have an ejector mechanism (the Fiskars tool, for example, has an ejector to push the pulled weed off the claws).

After successful removal, the robot reports the weed as handled (the weed's location could be marked "cleared" in the system). Finally, the robot will navigate back to where it left off in its coverage pattern or proceed to the next weed if multiple were detected. The ability to seamlessly transition from area coverage to a precise point-to-point task and then back is a highlight of the system's design. However, the working of the tool that was developed has a scope for improvement like refining and selecting an optimal motor torque as it was found that the motor torque for the rotation of the tool was insufficient.

7.6 Mowing Coordination and Blade Control

Mowing the lawn is a core function of the robot, and it is tightly integrated with the navigation system so that grass cutting occurs methodically and safely. The robot essentially performs mowing in tandem with its coverage path execution. Here's how the mowing functionality works:

Path-Integrated Mowing: The coverage path planning (described in section 3) is not only for finding weeds but also for mowing every inch of the lawn. The robot follows a predefined pattern (usually straight lines back and forth across the lawn) to cover the area. This pattern ensures overlap between passes so that no strips of grass are missed. For example, if the robot's cutting deck is 30 cm wide, the path lanes might be spaced slightly less than 30 cm apart to account for any slight deviations and ensure full coverage. The navigation stack handles driving these straight lines and making turns at the edges. Because the map has the lawn boundaries and obstacles, the robot only plans mowing paths within the lawn's area, avoiding flower beds, trees, or leaving the yard boundaries. This prevents it from accidentally mowing over unwanted areas.

Blade Activation and Control: The robot is equipped with a mowing blade underneath (similar to a traditional lawnmower blade) powered by an electric motor. This blade motor is controlled by the robot's onboard computer (through a motor driver). When the robot is about to start its coverage/mowing routine, it sends a command to activate the mower blade. In ROS2, this could be a simple service or topic (for instance, publishing an "ON" command to a /mower_blade topic). The blade spins up to the required speed (ensuring it has enough RPM to cut grass efficiently). As the robot drives along each lane, the blade continuously cuts the grass it passes over. The speed of the

robot is regulated – it moves at a steady, relatively slow mowing speed so that the blade has time to cut each blade of grass cleanly. If the robot moved too fast, it might not cut evenly, so the system might adjust speed based on grass thickness (in a basic setup, the speed is constant; in an advanced setup, current drawn by the blade motor could be monitored to detect heavy load and trigger a slowdown).

Uniform Cutting and Overlap: The goal is a uniform grass height across the lawn. The robot's blade height is set to a fixed level (e.g. 5 cm above ground) by the design via an adjustable mechanism. The mowing toolhead can be changed from the weeding toolhead as shown in Figure 7.10.

As it mows, it will trim all grass to that level. The overlapping path pattern ensures there are no uncut strips between lanes. If the robot's path planning and odometry are accurate, it will mow in neat lines, much like a human would, with a slight overlap on each pass to avoid any thin slivers of uncut grass. This produces a uniform cut pattern. Because the robot updates its position on the map continuously, it knows where it has been and where it has not, which helps in achieving complete coverage. Some systems even mark "already mowed" areas on a costmap so that the robot can focus on remaining areas.

Coordination and Safety: The blade control is coordinated with navigation such that if the robot needs to stop or divert (for instance, to avoid an obstacle or if a person/animal suddenly comes close), the blade can be stopped for safety. The robot has the ability to quickly shut off the blade if it's not moving or if a hazardous condition is detected (this prevents the blade from sitting and chewing into one spot or throwing debris). Also, when the robot finished a mowing session, it will turn off the blade. This is analogous to how a user would only engage the mower blades when actually cutting.

Figure 7.10: Mowing Toolhead

While the autonomous function of the lawn mowing system worked smoothly, the actual mowing performance did not meet the desired standards. There is clear potential for improvement in the tool's cutting effectiveness and precision. Future enhancements can address these shortcomings, ensuring that both the navigation and mowing tasks are executed with consistent quality throughout the process.

All the functional experiments explained above can be seen in the form of video through the link [69].

7.7 Summary

This chapter validates the performance of the custom-built autonomous lawn care robot, Dandebot, through real-world testing of its key functionalities. It begins with remote teleoperation, allowing safe and manual control during setup and emergencies. An intelligent SLAM-based mapping system, using LIDAR, IMU, and sensor fusion via an Extended Kalman Filter, enables the robot to localize and build accurate, real-time maps of outdoor environments. Complete

lawn coverage is achieved using the Fields2Cover library for path planning, enhanced by a custom behavior tree to handle dynamic obstacles. Detected weeds are logged and later targeted using precise point-to-point navigation and a mechanical extraction tool. Simultaneously, mowing is coordinated along the same paths using a dedicated blade control system. While the navigation and coordination systems performed reliably, improvements are needed in the weeding and mowing toolheads to enhance mechanical performance. Overall, the experiments confirmed that the robot can autonomously perceive, plan, and act in complex lawn environments.

Chapter 8

Conclusion and Future Directions

8.1 Conclusion

The aim of this thesis was to design and develop an autonomous robot capable of managing various aspects of lawn maintenance with minimal human intervention. Throughout the project, key problems related to weeding, mowing were addressed, while ensuring that the solution remains affordable and adaptable to different lawn conditions. The robot, named DandeBot, was developed through multiple stages, starting from initial designs to final testing of perception and control systems.

The project began with a study of the core problems in the lawn care industry and a review of existing solutions. From this, the need for innovation, especially in residential settings where manual labor and time costs are high, was identified. The literature review provided the foundation for understanding how robotics and automation, particularly mobile manipulators, can improve efficiency in outdoor maintenance tasks.

The kinematics and control of a differential drive robot were explored and validated using MATLAB simulations. Concepts such as polar coordinate transformations, Jacobian matrices, and linear control laws were applied to simulate and understand motion, which contributed to fine-tuning the physical behavior of the robot. Later, these concepts were implemented and tested on

the actual mobile platform.

The hardware design and evolution of the robot were also discussed, from early prototype versions (DandeBot v0.8 and v0.9) to the final base version (DandeBot v1.0). This included careful selection of the drivetrain, machining processes, and modular mechanisms that allow the robot to perform both mowing and weeding tasks. The mechanical structure was optimized for stability and ease of manufacturing.

In terms of electronics, several key components were integrated, such as the Jetson Orin Nano for processing, and controllers like Teensy 4.1 and ODrive v3.6 to manage motor control. Communication between modules was achieved using serial links, CAN bus, and peer-to-peer SSH. ROS 2 was used as the middleware to manage the perception, control, and navigation subsystems.

For perception, a custom dataset was collected using RC tyre wheels as placeholders for dandelions, and a YOLOv8 model was trained to detect and track these objects in real-time. The model achieved 97% accuracy with a frame rate of around 25 FPS, which proved reliable enough for navigation and action-based triggers. This detection was then connected to the control system to allow the robot to make decisions based on real-time input.

Finally, a two-layer control system was implemented—low-level motor control using cascaded PID loops and high-level navigation using the ROS 2 DWB local planner with multiple trajectory critics. Additional functionalities such as full lawn coverage, remote teleoperation, and an intelligent weed extraction mechanism were tested and integrated successfully.

This thesis brings all of these components together into a single working system. It demonstrates that, with the right hardware, software, and integration strategy, autonomous lawn care is not only possible but also practical. However, several areas remain where improvements could be made to bring the robot closer to being a commercial product.

8.2 Future Directions

Although the current prototype performs well in many conditions, several upgrades and extensions are planned for the future:

8.2.1 Aerating and Leaf Sweeping Integration:

One of the main goals moving forward is to expand the robot's functionality beyond mowing and weeding. Plans include adding aeration tools to improve soil health, as well as a leaf sweeping module to clean lawns during fall. These additional features will make the robot more versatile and capable of year-round lawn care.

8.2.2 Refinements and Software Upgrades:

While the current control and perception systems are functional, there is room for improvement in terms of efficiency, safety, and robustness. Future work will include refining the object detection pipeline to support real dandelion data, improving SLAM and localization performance in larger and more complex gardens, and making the system easier to update and maintain. Efforts will also focus on reducing CPU and GPU loads by optimizing code and removing unused nodes in the ROS architecture.

8.2.3 Finalizing the Embodiment Design for MVP (Minimum Viable Product):

The mechanical structure of the robot will be further refined to develop a compact, waterproof, and aesthetically pleasing design suitable for customer use. This will include better cable routing, weather-sealing of electronics, simplified assembly, and easier maintenance access. Weight reduction and optimized battery placement will also be considered to improve runtime and performance. The goal is to have a working MVP that is production-ready and can be tested in real backyards.

8.2.4 Testing with Real Dandelions and Customer Feedback:

In future stages, real-world testing will be conducted using actual lawn conditions and dandelion species. This will help fine-tune the weeding mechanism and improve AI performance. Feedback will also be collected from homeowners to understand what features are most useful and what needs to be improved for better user experience.

The futuristic all in one commercial lawn care robot can be seen in Figure 8.1.

Figure 8.1: Future Concept LawnCar

In summary, while this thesis presents a strong foundation, the journey of DandeBot does not end here. With the planned additions and refinements, it is believed that this robot can become a fully autonomous and intelligent lawn care system that will eventually replace traditional methods in an eco-friendly and cost-effective way.

References

[1] Fiskars. *Weeding Tools*. https://www.fiskars.com/en/gardening/product-categories/weeding-tools. Accessed: 2025-03-25. 2025.

[2] Jim Bowyer et al. "Environmental Assessment of Intensive Lawn Care". In: *Dovetail Partners, Inc.* (2019).

[3] W. J. Hundertmark, M. Lee, and I. A. Smith. "Influence of landscape management practices on urban greenhouse gas budgets". In: *Carbon Balance and Management* 16.1 (2021), pp. 1–15.

[4] Tracy L. Fuentes. "Homeowner preferences drive lawn care practices and species diversity patterns in new lawn floras". In: *Journal of Urban Ecology* 7.1 (2021), juab015.

[5] Maria Ignatieva et al. "Lawn as a cultural and ecological phenomenon: A conceptual framework for transdisciplinary research". In: *Urban Forestry & Urban Greening* 14.2 (2015), pp. 383–387. DOI: 10.1016/j.ufug.2015.04.003.

[6] N.F. Martini and K.C. Nelson. "The role of knowledge in residential lawn management". In: *Urban Ecosystems* 18.5 (2015), pp. 1031–1047.

[7] Michael Graef et al. "Sustaining Green: Quality Improvement of Green Infrastructure in Residential Facilities through Effective Maintenance and Resident Participation". In: *Journal of Facilities Management* 25.1 (2023), pp. 46–59. DOI: 10.34749/jfm.2023.4667.

[8] Statista Research Department. *U.S. household expenditure on gardening/lawn care services 2007–2022*. https://www.statista.com/statistics/468171/us-consumer-spending-on-gardening-lawn-care-services/. Accessed: 2025-03-25. 2022.

[9] ROI Revolution. *2021 Lawn & Garden Industry Deep Dive: Data, Stats, & Trends for Brands*. https://roirevolution.com/blog/2021-lawn-

garden-industry-deep-dive-data-stats-trends-for-brands/. Accessed: 2025-03-25. 2021.

[10] Statista Research Department. *Annual household expenditure on garden supplies and services in Canada from 2010 to 2021*. https://www.statista.com/statistics/436444/annual-household-expenditure-on-garden-supplies-services-canada/. Accessed: 2025-03-25. 2021.

[11] Exmerce. *10 Things for Summer You Can Get on Barter*. https://exmerce.com/10-things-for-summer-you-can-get-on-barter/. Accessed: 2025-03-25. 2023.

[12] LawnStarter. *Time Spent on Lawn Care, Gardening Hits Low, Then Surges*. https://www.lawnstarter.com/blog/studies/time-spent-on-lawn-care-gardening-hits-low-then-surges/. Accessed: 2025-03-25. 2019.

[13] GreenPal. *Survey: How Many Hours a Week Do Homeowners Spend on Lawn Care?* https://www.yourgreenpal.com/blog/survey-homeowners-weekly-lawn-care-hours. Accessed: 2025-03-25. 2023.

[14] Angi. *How Much Does Lawn Care Cost?* https://www.angi.com/articles/lawn-care-cost.htm. Accessed: 2025-03-25. 2023.

[15] FieldCamp. *Top Lawn Care Industry Statistics and Trends for 2023*. https://fieldcamp.ai/blog/lawn-care-industry-statistics-and-trends/. Accessed: 2025-03-25. 2023.

[16] My Backyard Compost Pile. *The Lawn: A History of an American Obsession*. https://mybackyardcompostpile.com/tag/the-lawn-a-history-of-an-american-obsession/. Accessed: 2025-03-25. 2023.

[17] Business Wire. *Robotic Lawn Mower Market Projected to Experience Robust Growth from 2024–2029, Driven by Green Space Adoption and Cost Efficiency – ResearchAndMarkets.com*. https://www.businesswire.com/news/home/20240320319785/en/Robotic-Lawn-Mower-Market-Projected-to-Experience-Robust-Growth-from-2024-2029-Driven-by-Green-Space-Adoption-and-Cost-Efficiency---ResearchAndMarkets.com. Accessed: 2025-03-25. 2024.

[18] CNET. *Are Robot Lawn Mowers Worth the Money Yet?* https://www.cnet.com/home/are-robot-lawn-mowers-worth-the-money-yet/. Accessed: 2025-03-25. 2023.

[19] Ritwik PK and Nishigandha Patel. “Autonomous Lawn Mower 2013 A Comprehensive Review”. In: *International Research Journal on Advanced Science Hub* 05.12 (2023), pp. 420–428. DOI: 10.47392/IRJASH.2023.079.

[20] Tafadzwa Abbas et al. “Precision Weed Management Using Autonomous Technologies”. In: *Agricultural Robotics and Sustainable Development* 34.2 (2018), pp. 112–125. DOI: 10.1016/j.agsy.2018.01.009.

[21] Nathan T. Lunstad and Robert B. Sowby. “Smart Irrigation Controllers in Residential Applications and the Potential of Integrated Water Distribution Systems”. In: *Journal of Water Resources Planning and Management* 150.1 (2024), p. 03123002. DOI: 10.1061/JWRMD5.WRENG-5871. URL: https://doi.org/10.1061/JWRMD5.WRENG-5871.

[22] Ata Jahangir Moshayedi et al. “Robots in Agriculture: Revolutionizing Farming Practices”. In: *EAI Endorsed Transactions on AI and Robotics* (June 2024). DOI: 10.4108/airo.5855.

[23] Iron Solutions. *The History of Lawn Mowers*. https://ironsolutions.com/the-history-of-lawn-mowers/. Accessed: 2025-03-25. 2025.

[24] Wikipedia contributors. *Lawn mower*. https://en.wikipedia.org/wiki/Lawn_mower. Accessed: 2025-03-25. 2025.

[25] C. Cheng et al. “Recent Advancements in Agriculture Robots: Benefits and Challenges”. In: *Machines* 11.1 (2023), p. 48. DOI: 10.3390/machines11010048.

[26] GlobeNewswire. *Robotic Lawn Mowers Market Trends and Business Strategies: Forecast to 2030*. https://www.globenewswire.com/news-release/2024/12/11/2995233/28124/en/Robotic-Lawn-Mowers-Market-Trends-and-Business-Strategies-Forecast-to-2030.html. Accessed: 2025-03-25. 2024.

[27] PR Newswire. *Robotic Lawn Mower Market Strategic Assessment 2024–2029: Increasing Adoption of Green Spaces & Green Roofs Powers Market Growth – CAGR of 11.35% Forecast During 2023–2029*. https://www.prnewswire.com/news-releases/robotic-lawn-mower-market-strategic-assessment-2024-2029-increasing-adoption-of-green-spaces--green-roofs-powers-market-growth---cagr-of-11-35-forecast-during-2023-2029--302076197.html. Accessed: 2025-03-25. 2024.

[28] RobotsGuide. *Tertill*. https://robotsguide.com/robots/tertill. Accessed: 2025-03-25. 2023.

[29] NC State Extension. *Artificial Intelligence (AI)-Enabled Robotic Weeders in Precision Agriculture*. https://content.ces.ncsu.edu/artificial-intelligence-ai-enabled-robotic-weeders-in-precision-agriculture. Accessed: 2025-03-25. 2023.

[30] M. Wakchaure, B.K. Patle, and A.K. Mahindrakar. "Application of AI Techniques and Robotics in Agriculture: A Review". In: *Artificial Intelligence in the Life Sciences* 3 (2023), p. 100057. DOI: 10.1016/j.ailsci.2023.100057.

[31] Husqvarna. *Automower 435X AWD*. Accessed: 2025-04-20. 2024. URL: https://www.husqvarna.com/ca-en/robotic-lawn-mowers/automower-435x-awd/.

[32] Wikipedia contributors. *Robotic lawn mower*. https://en.wikipedia.org/wiki/Robotic_lawn_mower. Accessed: 2025-03-25. 2023.

[33] Laidback Gardener. *Robotic Lawn Mowers Come Out of the Shadows*. https://laidbackgardener.blog/2017/05/02/robotic-lawn-mowers-come-out-of-the-shadows/. Accessed: 2025-03-25. 2017.

[34] Easy Lawn Mowing. *Small Problem with Robotic Mowers: Setting Up and Installation*. https://easylawnmowing.co.uk/small-problem-with-roboticmowers/#\:~:text=Small\Problem\with\Robotic\Mowers%3A,Setting\Up\:\Installation. Accessed: 2025-03-25. 2023.

[35] PCMag. *The Best Robot Lawn Mowers*. https://www.pcmag.com/picks/the-best-robot-lawn-mowers. Accessed: 2025-03-25. 2023.

[36] Husqvarna Research. *HRP - Husqvarna Research Projects*. https://github.com/HusqvarnaResearch/hrp. Accessed: 2025-03-25. 2025.

[37] MDPI. "The Paper Discusses the State of the Art in Robotic Taxonomies". In: *MDPI* 10.8 (2021), p. 648. DOI: 10.3390/2075-1702/10/8/648. URL: https://www.mdpi.com/2075-1702/10/8/648.

[38] Belrobotics. *Bigmow Connected Line*. https://www.belrobotics.com/en/robotic-mowers/bigmow-connected-line/. Accessed: 2025-03-25. 2023.

[39] IEEE Spectrum. *iRobot Terra Robotic Lawnmower: Preview and Insights*. https://spectrum.ieee.org/irobot-terra-robotic-lawnmower. Accessed: 2025-03-25. 2023.

[40] Yarbo. *Lawn Mower Robot*. https://www.yarbo.com/products/lawn-mower-robot. Accessed: 2025-03-25. 2023.

[41] Robotics Stack Exchange. *Wheels vs Continuous Tracks (Tank Treads)*. https://robotics.stackexchange.com/questions/541/wheels-vs-continuous-tracks-tank-treads. Accessed: 2025-03-25. 2023.

[42] Clearpath Robotics. *Husky A300 Unmanned Ground Vehicle Robot*. https://clearpathrobotics.com/husky-a300-unmanned-ground-vehicle-robot/. Accessed: 2025-03-25. 2024.

[43] ODrive Robotics. *Control Manual*. Accessed: 2025-04-20. 2024. URL: https://docs.odriverobotics.com/v/latest/manual/control.html.

[44] NVIDIA. *Autonomous Machines Embedded Systems*. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. Accessed: 2025-03-25. 2024.

[45] Circuit Digest. *ESP32 Self Driving Robot with Arduino, ROS2 and LIDAR*. https://circuitdigest.com/news/esp32-self-driving-robot-with-arduino-ros2-lidar. Accessed: 2025-03-25. 2024.

[46] Concordia University. *Approach Developed to Address Dandelion Detection and Recognition*. https://spectrum.library.concordia.ca/id/eprint/987195/. Accessed: 2025-03-25. 2024.

[47] SSRN. *Relative Robustness to Noise and Its Use in the Landscaping Industry*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4693544. Accessed: 2025-03-25. 2024.

[48] MDPI. "YOLO: A Comprehensive Overview and the Latest Updates". In: *Sensors* 24.13 (2024). Accessed: 2025-03-25, p. 4379. DOI: 10.3390/s24134379. URL: https://www.mdpi.com/1424-8220/24/13/4379.

[49] Pace Recruiters. *How to Get ROS Up and Running on Spot*. https://pacerecruiters.com/robotics-engineering-jobs-boston/how-to-get-ros-up-running-on-spot. Accessed: 2025-03-25. 2020.

[50] DFRobot. *Brief Analysis of Camera Principles*. Accessed: 2025-04-20. 2023. URL: https://wiki.dfrobot.com/brief_analysis_of_camera_principles.

[51] FLIR Systems. *FLIR Systems Stereo Cameras Enable Autonomous Mobile and Pick-and-Place Robots*. Accessed: 2025-04-20. 2023. URL: https://www.automationworld.com/factory/sensors/article/55271683/

flir-systems-stereo-cameras-enable-autonomous-mobile-and-pick-and-place-robots.

[52] RealPars. *Limit Switch*. Accessed: 2025-04-20. 2023. URL: https://www.realpars.com/blog/limit-switch.

[53] Wikipedia contributors. *Incremental Encoder*. Accessed: 2025-04-20. 2023. URL: https://en.wikipedia.org/wiki/Incremental_encoder.

[54] Battle Born Batteries. *What is a Battery Monitor?* Accessed: 2025-04-20. 2023. URL: https://battlebornbatteries.com/what-is-a-battery-monitor.

[55] Real Pars. *Servo Motor*. Accessed: 2025-04-20. 2023. URL: https://www.realpars.com/blog/servo-motor.

[56] Wikipedia contributors. *Stepper Motor*. Accessed: 2025-04-20. 2023. URL: https://en.wikipedia.org/wiki/Stepper_motor.

[57] Aspina Group. *What Is*. Accessed: 2025-04-20. 2023. URL: https://us.aspina-group.com/en/learning-zone/columns/what-is/013/.

[58] InfoQ. *NVIDIA Jetson AI Supercomputer*. Accessed: 2025-04-20. 2024. URL: https://www.infoq.com/news/2024/12/nvidia-jetson-ai-supercomputer/.

[59] SparkFun. *Teensy 4.1*. Accessed: 2025-04-20. 2024. URL: https://www.sparkfun.com/teensy-4-1.html.

[60] ODrive Robotics. *ODrive v3.6*. Accessed: 2025-04-20. 2024. URL: https://shop.odriverobotics.com/products/odrive-v36.

[61] Reprap Forum. *Hey Reprappers, I would just*. Accessed: 2025-04-20. 2024. URL: https://reprap.org/forum/read.php?1,661600.

[62] Newhaven Display. *Serial vs Parallel Communication*. Accessed: 2025-04-20. 2024. URL: https://newhavendisplay.com/blog/serial-vs-parallel-communication/.

[63] CSSelectronics. *CAN Bus Simple Intro Tutorial*. Accessed: 2025-04-20. 2024. URL: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial.

[64] GeeksforGeeks. *Robot Operating System (ROS): The Future of Automation*. Accessed: 2025-04-20. 2024. URL: https://www.geeksforgeeks.org/robot-operating-system-ros-the-future-of-automation.

[65] Peter Allen. *Inverse Kinematics*. https://www.cs.columbia.edu/~allen/F17/NOTES/icckinematics.pdf. Accessed: 2025-03-25. 2017.

[66] Nav2. *Configuring DWB Controller*. Accessed: 2025-04-20. 2024. URL: https://docs.nav2.org/configuration/packages/configuring-dwb-controller.html.

[67] ROS Navigation. *DWB Controller README*. Accessed: 2025-04-20. 2024. URL: https://github.com/ros-navigation/navigation2/blob/main/nav2_dwb_controller/README.md.

[68] ROS2 Documentation. *Base Obstacle Critic in ROS2 Navigation*. Accessed: 2025-04-20. 2024. URL: https://www.ncnynl.com/ros2docs/en/nav2/configuration/packages/trajectory_critics/base_obstacle.html.

[69] Google Drive. *Shared Folder*. https://drive.google.com/drive/folders/1am_h_GcwrEGqDxCvFTTi-zTnd_3_NDgb?usp=sharing. Accessed: 2025-03-25. 2025.

Appendix A

Forward Kinematic Motion

```
clear all; close all; clc;
dt=1;ts=25;t=0:dt:ts;
x0=0.2;  y0=0.2; psi0=pi/4;
a=0.05;  d=0.1;
eta0 = [x0;y0;psi0];
eta(:,1) = eta0;
omega1=1; omega2=0.5;
for i=1:length(t)
    psi=eta(3,i);

    J_psi=[cos(psi),-sin(psi),0;
           sin(psi),cos(psi),0;
           0,0,1];
       W=[a/2,a/2;
          0,0;
          (-a/(2*d)),(a/(2*d))];
      omega=[omega1;omega2];
    zeta(:,i)=W*omega;
    eta_dot(:,i)=J_psi*zeta(:,i);
    eta(:,i+1)=eta(:,i)+dt*eta_dot(:,i);
end
    l=2*d; w=0.3;
    mr_co=[-l/2,l/2,l/2,-l/2;
       -w/2,-w/2,w/2,w/2
       0,0,0,0];
figure
for i=1:5:length(t)
    psi=eta(3,i);
```

```
29      R_psi=[cos(psi),-sin(psi),0;
30             sin(psi),cos(psi),0;
31             0,0,1];
32      V_pos=R_psi*mr_co;
33      fill(V_pos(1,:)+eta(1,i),V_pos(2,:)+eta(2,i),'g')
34      hold on, grid on
35      axis([-1 3 -1 3]), axis square
36      plot(eta(1,1:i),eta(2,1:i),'b-');
37      legend('Robot','Path')
38      set(gca,'fontsize',24);
39      xlabel('x,[m]');
40      ylabel('y,[m]');
41      pause(0.01);
42      hold off
43  end
```

Listing A.1: Forward Kinematic Motion

Appendix B

Inverse Kinematic Polar Coordinates Control

```
1
2  clear all;
3  close all;
4  clc;
5
6  %%Time
7  dt=0.01; ts=5; t=0:dt:ts;
8
9  %%Error Circle Radius
10 r = 0.1;
11
12 %%Initial Position
13 x0=0;
14 y0=0;
15 psi0=pi/4;
16
17 %%Dimensions of the Robot
18 a=0.05;
19 d=0.2;
20 l=1.4;
21
22 %%Desired Position
23 xd = 10;
24 yd = 5;
25 psid = 0;
26 gcd  = [xd;yd;psid]
```

```
27
28
29  eta = zeros(3,length(t));
30  eta_dot = zeros(3,length(t));
31  gc_dot = zeros(3,length(t));
32
33  %gc0 = zeros(3,length(t));
34  for i=1:length(t)
35
36      gc_d(1,i)=x0;
37      gc_d(2,i)=y0;
38      gc_d(3,i)=psi0;
39  end
40
41  %%Initial positions append to the array
42  gc(:,1) = [x0;y0;psi0];
43
44  %%Polar Coordinate Constant Gains
45  Kp = 3;
46  Ka = 8;
47  Kb = -1.5;
48
49  eta = zeros(3, length(t));
50  for i=1:length(t)
51
52      %%Finding (DEL X,Y),Psi
53      delx = xd-gc(1,i);
54      dely = yd-gc(2,i);
55      psi = gc(3,i);
56
57      %%Finding Rho,Alpha,Beta
58      rho = sqrt(delx^2+dely^2);
59      alpha = -psi + atan2(dely,delx);
60      beta = - psi - alpha ;
61
62      %%Initializing initial values
63      eta(:,i) = [rho;alpha;beta];
64
65      %%Jacobian Matrix
66      J = [-cos(alpha),0;(sin(alpha)/rho),-1;(-sin(alpha)/rho),0];
67
68      %%Relation Between Zeta and Polar Coordinates
69      zeta = [Kp*rho;Ka*alpha+Kb*beta];
```

```
70
71      %%velocity in Polar Coordinate form
72      eta_dot(:,i) = [-Kp,0,0;0,-(Ka-Kp),-Kb;0,-Kp,0]*eta(:,i);
73
74      %%Using Euler's Approximation
75      eta(:,i+1) = eta(:,i) + eta_dot(:,i)*dt;
76
77      %%Convert back to Cartesian Plane (Ig this is wrong)
78       J1 = [cos(psi),0;sin(psi),0;0,1];
79       gc_dot(:,i) = J1*pinv(J)*eta_dot(:,i);
80       gc(:,i+1) = gc(:,i) + gc_dot(:,i)*dt;
81  end
82  %%Animation
83  width = 1.2; % width of the mobile robot
84  veh_box = 0.5*[-l/2,l/2,l/2,-l/2,-l/2;
85          -width/2,-width/2,width/2,width/2,-width/2;];
86  cas_p = 0.2*[0.6;0];
87  wheel_b = 0.5*[-0.2 0.2 0.2 -0.2 -0.2;-0.05 -0.05 0.05 0.05 -0.05];
88  for i = 1:length(t)
89      psi = gc(3,i);
90      x(i) = gc(1,i);
91      y(i) = gc(2,i);
92      R = [cos(psi),-sin(psi);sin(psi),cos(psi)];
93      v_m = R*veh_box;
94      c_m = R*cas_p;
95      w_m1 = R*(wheel_b+[0;0.35/2]);
96      w_m2 = R*(wheel_b+[0;-0.35/2]);
97
98      fill(v_m(1,:)+x(i),v_m(2,:)+y(i),'y');
99      hold on
100     fill(w_m1(1,:)+x(i),w_m1(2,:)+y(i),'r');
101     fill(w_m2(1,:)+x(i),w_m2(2,:)+y(i),'b');
102     fill(c_m(1)+0.05*cosd(0:360)+x(i),c_m(2)+0.05*sind(0:360)+y(i),'g');
103
104     plot(gc(1,1:i),gc(2,1:i),'b-');
105     plot(gcd(1),gcd(2),'k*')
106     plot([gcd(1),gcd(1)+2*r*cos(gcd(3))],[gcd(2),gcd(2)+2*r*sin(gcd(3))]) ↩
            ;
107     plot(gcd(1)+r*cosd(0:360),gcd(2)+r*sind(0:360),'c--')
108
109     margin = 1.5;
110     xmin = min([gc(1,:), gc_d(1)]) - margin;
111     xmax = max([gc(1,:), gc_d(1)]) + margin;
```

```
112     ymin = min([gc(2,:), gc_d(2)]) - margin;
113     ymax = max([gc(2,:), gc_d(2)]) + margin;
114     axis([xmin xmax ymin ymax])
115 
116     axis equal
117     grid on
118     xlabel('x,[m]')
119     ylabel('y,[m]')
120     pause(0.01)
121     hold off
122 end
123 
124 % Plot for Movement in Cartesian Plane
125 figure
126 
127 subplot(2,1,1)
128 plot(t, repmat(gcd(1), length(t), 1),'r-') % Repeat the constant value ↩
        for all t
129 hold on
130 plot(t, gc(1,1:i), 'b-')
131 legend('Desired', 'Actual');
132 xlabel('Time')
133 ylabel('X Position')
134 
135 subplot(2,1,2)
136 plot(t, repmat(gcd(2), length(t), 1),'r-') % Repeat the constant value ↩
        for all t
137 hold on
138 plot(t, gc(2,1:i), 'b-')
139 legend('Desired', 'Actual');
140 xlabel('Time')
141 ylabel('Y Position')
```

Listing B.1: Inverse Kinematic Polar Coordinates Control

Appendix C

ROSSerial Arduino with Kinematic Conversion

```
#include <Encoder.h>
#include <PID_v1.h>
#include <ros.h>
#include <Stepper.h>
#include <tf/transform_broadcaster.h>
#include <geometry_msgs/TransformStamped.h>
#include <geometry_msgs/Twist.h>
#include <geometry_msgs/PoseWithCovarianceStamped.h>
#include <nav_msgs/Odometry.h>
#include <geometry_msgs/Vector3.h>
#include <tf/tf.h>
#include <ros/time.h>
#include <actionlib_msgs/GoalStatusArray.h>
#include <std_msgs/UInt8.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>
#include <sensor_msgs/Imu.h>
#include <geometry_msgs/Vector3Stamped.h>

#define IN1 8
#define IN2 6
#define PWM1 7 //LEFT
#define IN3 9
#define IN4 11
```

```
27 #define PWM2 10 //RIGHT
28 #define IN5 13
29 #define IN6 15
30 #define PWM3 14 //ACTUATOR
31 #define LIMITB 30
32 #define LIMITT 29
33
34
35 double Pk = 50;
36 double Ik = 80;
37 double Dk = 0.3;
38 double Pk1 = 50;
39 double Ik1 = 80;
40 double Dk1 = 0.3;
41
42 double theta = 0;
43 char base_link[] = "/base_link";
44 char odom[] = "/odom";
45
46 bool rotateClockwise = false;
47 bool rotateCounterclockwise = false;
48
49 float demand1, demand2, demandx, demandz, LeftOutput, RightOutput;
50 const int stepsPerRevolution = 1000;
51 Stepper myStepper(stepsPerRevolution, 12, 39, 28, 40);
52
53 ros::NodeHandle nh;
54 geometry_msgs::TransformStamped t;
55 nav_msgs::Odometry odom_msg;
56 tf::TransformBroadcaster broadcaster;
57 std_msgs::UInt8 StepperStatus;
58 sensor_msgs::Imu imu_msg;
59 ros::Publisher imu_pub("imu/data", &imu_msg);
60 geometry_msgs::Vector3Stamped euler_msg;
61 geometry_msgs::Vector3Stamped gyro_msg;
62 geometry_msgs::Vector3Stamped linear_accel_msg;
63
64 Adafruit_BNO055 bno = Adafruit_BNO055(55, 0x28, &Wire);
65
66 double orientation_covariance[9] = {0.0025, 0, 0, 0, 0.0025, 0, 0, 0,  ↩
       0.0025};
67 double angular_velocity_covariance[9] = {0.02, 0, 0, 0, 0.02, 0, 0, 0,  ↩
       0.02};
```

```
68  double linear_acceleration_covariance[9] = {0.04, 0, 0, 0, 0.04, 0, 0,  ↩
        0, 0.04};
69
70  void onTwist(const geometry_msgs::Twist& msg) {
71    demandx = msg.linear.x;
72    demandz = msg.angular.z;
73  }
74
75  ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", onTwist);
76  ros::Subscriber<actionlib_msgs::GoalStatusArray> sub1("move_base/status" ↩
        , statusCallback);
77  ros::Publisher odom_pub("odom", &odom_msg);
78  ros::Publisher stepperStatusPub("stepper_status", &StepperStatus);
79  bool taskCompleted = false;
80
81  void statusCallback(const actionlib_msgs::GoalStatusArray &msg) {
82    if (msg.status_list_length > 0) {
83      int status = msg.status_list[0].status;
84
85      if (status == actionlib_msgs::GoalStatus::SUCCEEDED && taskCompleted  ↩
            == false) {
86        // Goal reached, perform the stepper motor task
87        rotateCounterclockwise = true;
88        rotateClockwise = false;
89        StepperStatus.data = 0;
90        stepperStatusPub.publish(&StepperStatus);
91        taskCompleted = true;
92      } else if (status == actionlib_msgs::GoalStatus::ACTIVE) {
93        // A new goal is being processed, reset the taskCompleted flag
94        taskCompleted = false;
95      }
96    }
97  }
98
99
100
101 double Setpoint, Input, Output, Outputa;
102 PID PID1(&Input, &Output, &Setpoint, Pk, Ik, Dk, DIRECT);
103 double Setpoint1, Input1, Output1, Outputa1;
104 PID PID2(&Input1, &Output1, &Setpoint1, Pk1, Ik1, Dk1, DIRECT);
105
106 bool resetInProgress = true;
107
```

```
int buzzer = 36;

Encoder knobLeft(5, 4);
Encoder knobRight(2, 3);
double currentMillis, previousMillis, Interval;


void setup() {
  //Serial.begin(115200);
  //Serial.setTimeout(10);
  pinMode(buzzer, OUTPUT);
  tone(buzzer, 1000);
  delay(500);
  noTone(buzzer);
  delay(500);
  tone(buzzer, 1000);
  delay(500);
  noTone(buzzer);
  delay(500);
  PID1.SetMode(AUTOMATIC);
  PID1.SetOutputLimits(-255, 255);
  PID1.SetSampleTime(10);
  PID2.SetMode(AUTOMATIC);
  PID2.SetOutputLimits(-255, 255);
  PID2.SetSampleTime(10);
  pinMode(IN4, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(PWM2, OUTPUT);
  pinMode(PWM1, OUTPUT);
  pinMode(IN5, OUTPUT);
  pinMode(IN6, OUTPUT);
  pinMode(PWM3, OUTPUT);
  pinMode(LIMITT, INPUT_PULLUP);
  pinMode(LIMITB, INPUT_PULLUP);

  myStepper.setSpeed(1000);


  nh.initNode();
  nh.subscribe(sub);
  nh.advertise(odom_pub);
```

```
151   nh.subscribe(sub1);
152   nh.advertise(stepperStatusPub);
153   nh.advertise(imu_pub);
154   if (!bno.begin()) {
155     while (1);
156   }
157   bno.setExtCrystalUse(true);
158   broadcaster.init(nh);
159 
160 }
161 
162 long positionLeft = 0;
163 long positionRight = 0;
164 double distx = 0;
165 double disty = 0;
166 double deltaX = 0;
167 double deltaY = 0;
168 int timee = 0;
169 double pos_total_mm = 0;
170 float RPMLeftPrev = 0;
171 float LeftFilt = 0;
172 
173 float RPMRightPrev = 0;
174 float RightFilt = 0;
175 
176 
177 
178 void loop() {
179   nh.spinOnce();
180   currentMillis = millis();
181 
182   int TopSwitch = digitalRead(LIMITT);
183   int BottomSwitch = digitalRead(LIMITB);
184 
185   if (currentMillis - previousMillis > 10) {
186 
187     sensors_event_t angVelocityData , linearAccelData;
188     bno.getEvent(&angVelocityData, Adafruit_BNO055::VECTOR_GYROSCOPE);
189     bno.getEvent(&linearAccelData, Adafruit_BNO055::VECTOR_LINEARACCEL);
190 
191     printEvent(&angVelocityData);
192     printEvent(&linearAccelData);
193 
```

```
194     imu::Quaternion quat = bno.getQuat();
195
196     if (resetInProgress) {
197       // Move the stepper motor up (counterclockwise) until TopSwitch is ↩
              activated
198       rotateClockwise = true;
199       rotateCounterclockwise = false;
200       tone(buzzer, 500);
201
202
203       // Check if the TopSwitch is activated, if yes, reset is complete
204       if (TopSwitch == 0) {
205         // Set the reset flag to false to indicate reset is complete
206         resetInProgress = false;
207         rotateClockwise = false;
208         rotateCounterclockwise = false;
209         myStepper.step(0);
210         noTone(buzzer);
211
212       }
213     }
214
215     if (rotateClockwise) {
216       myStepper.step(stepsPerRevolution);
217       tone(buzzer, 1000);
218       //noTone(buzzer);
219       if (TopSwitch == 0) {
220         Serial.println("Hit The Top Limit");
221         rotateClockwise = false;
222         rotateCounterclockwise = false;
223         noTone(buzzer);
224         StepperStatus.data = 1;
225         stepperStatusPub.publish(&StepperStatus);
226       }
227     }
228
229     if (rotateCounterclockwise) {
230       myStepper.step(-stepsPerRevolution);
231       tone(buzzer, 500);
232       //noTone(buzzer);
233       if (BottomSwitch == 0) {
234         Serial.println("Hit the Bottom Limit");
235         rotateClockwise = true;
```

```
236         rotateCounterclockwise = false;
237         noTone(buzzer);
238       }
239     }
240
241     Interval = currentMillis - previousMillis;
242     double newLeft, newRight, LeftDiff, RightDiff, LeftDiffmm,  ↩
            RightDiffmm, pos_average_mm_diff;
243     //Serial.println(Interval);
244     double LeftPPS, RightPPS, DistEachPulse, LinLeft, LinRight, RPMLeft,  ↩
            RPMRight;
245     newLeft = knobLeft.read();
246     newRight = knobRight.read();
247
248     LeftDiff = newLeft - positionLeft;
249     RightDiff = newRight - positionRight;
250
251     LeftPPS = LeftDiff * 1000 / Interval;
252     RightPPS = RightDiff * 1000 / Interval;
253
254     DistEachPulse = 0.0002;
255
256     LinLeft = LeftPPS * DistEachPulse;
257     LinRight = RightPPS * DistEachPulse;
258
259     RPMLeft = (LinLeft * 60) / (2 * 3.14 * 0.121);
260     RPMRight = (LinRight * 60) / (2 * 3.14 * 0.121);
261
262     LeftFilt = 0.93908194 * LeftFilt + 0.03045903 * RPMLeft + 0.03045903  ↩
            * RPMLeftPrev;
263     RPMLeftPrev = RPMLeft;
264
265     RightFilt = 0.93908194 * RightFilt + 0.03045903 * RPMRight +  ↩
            0.03045903 * RPMRightPrev;
266     RPMRightPrev = RPMRight;
267
268     positionLeft = newLeft;
269     positionRight = newRight;
270
271     demand1 = (demandx - (demandz * 0.364)); //0.73/2
272     demand2 = (demandx + (demandz * 0.364));
273
274     LeftOutput = (demand1 * 60) / (2 * 3.14 * 0.121);
```

```
275     RightOutput = (demand2 * 60) / (2 * 3.14 * 0.121);
276
277     Setpoint = LeftOutput;
278     Setpoint1 = RightOutput;
279
280     Input = LeftFilt;
281     Input1 = RightFilt;
282
283     PID1.Compute();
284     PID2.Compute();
285
286     if (Output > 0) {
287       if (Output1 > 0) {
288         Outputa = abs(Output); //LEFT
289         Outputa1 = abs(Output1); //RIGHT
290         digitalWrite(IN1, HIGH);
291         digitalWrite(IN2, LOW);
292         analogWrite(PWM1, Outputa);
293         digitalWrite(IN3, HIGH);
294         digitalWrite(IN4, LOW);
295         analogWrite(PWM2, Outputa1);
296       } else if (Output1 < 0) {
297         Outputa = abs(Output); //LEFT
298         Outputa1 = abs(Output1); //RIGHT
299         digitalWrite(IN1, HIGH);
300         digitalWrite(IN2, LOW);
301         analogWrite(PWM1, Outputa);
302         digitalWrite(IN3, LOW);
303         digitalWrite(IN4, HIGH);
304         analogWrite(PWM2, Outputa1);
305       }
306     } else if (Output < 0) {
307       if (Output1 > 0) {
308         Outputa = abs(Output); //LEFT
309         Outputa1 = abs(Output1); //RIGHT
310         digitalWrite(IN1, LOW);
311         digitalWrite(IN2, HIGH);
312         analogWrite(PWM1, Outputa);
313         digitalWrite(IN3, HIGH);
314         digitalWrite(IN4, LOW);
315         analogWrite(PWM2, Outputa1);
316       } else if (Output1 < 0) {
317         Outputa = abs(Output); //LEFT
```

```
318         Outputa1 = abs(Output1); //RIGHT
319         digitalWrite(IN1, LOW);
320         digitalWrite(IN2, HIGH);
321         analogWrite(PWM1, Outputa);
322         digitalWrite(IN3, LOW);
323         digitalWrite(IN4, HIGH);
324         analogWrite(PWM2, Outputa1);
325       }
326     }
327
328     LeftDiffmm = LeftDiff / 6.32;
329     RightDiffmm = RightDiff / 6.32;
330
331     // calc distance travelled based on average of both wheels
332     pos_average_mm_diff = (LeftDiffmm + RightDiffmm) / 2;
333     pos_total_mm += pos_average_mm_diff; // calc total running total ↩
          distance
334
335     // calc angle or rotation to broadcast with tf
336     theta += (RightDiffmm - LeftDiffmm) / (360 * 3.53);
337
338     if (theta > PI)
339       theta -= TWO_PI;
340     if (theta < (-PI))
341       theta += TWO_PI;
342
343     // calc x and y to broadcast with tf
344
345     distx += pos_average_mm_diff * cos(theta);
346     disty += pos_average_mm_diff * sin(theta);
347
348     geometry_msgs::TransformStamped t;
349     t.header.stamp = nh.now();
350     t.header.frame_id = "odom";
351     t.child_frame_id = "base_link";
352     t.transform.translation.x = distx / 1000; // convertto metres
353     t.transform.translation.y = disty / 1000;
354     t.transform.translation.z = 0;
355     t.transform.rotation = tf::createQuaternionFromYaw(theta);
356     broadcaster.sendTransform(t);
357
358
359     nav_msgs::Odometry odom_msg;
```

```
360
361     odom_msg.header.stamp = nh.now();
362     odom_msg.header.frame_id = "odom";
363     odom_msg.pose.pose.position.x = distx / 1000;
364     odom_msg.pose.pose.position.y = disty / 1000;
365     odom_msg.pose.pose.position.z = 0.0;
366     odom_msg.pose.pose.orientation = tf::createQuaternionFromYaw(theta);
367
368     odom_msg.child_frame_id = "base_link";
369     odom_msg.twist.twist.linear.x = ((LeftDiffmm + RightDiffmm) / 2) /  ↩
          10; // Linear Velocity
370     odom_msg.twist.twist.linear.y = 0.0; // robot does not move sideways
371     odom_msg.twist.twist.angular.z = ((LeftDiffmm - RightDiffmm) / (360 * ↩
           3.53)) * 100;
372
373
374
375     // Fill in the IMU message
376     imu_msg.header.stamp = nh.now();
377     imu_msg.header.frame_id = "imu_link";
378     imu_msg.orientation.w = quat.w();
379     imu_msg.orientation.x = quat.x();
380     imu_msg.orientation.y = quat.y();
381     imu_msg.orientation.z = quat.z();
382     imu_msg.angular_velocity.x = gyro_msg.vector.x;
383     imu_msg.angular_velocity.y = gyro_msg.vector.y;
384     imu_msg.angular_velocity.z = gyro_msg.vector.z;
385     imu_msg.linear_acceleration.x = linear_accel_msg.vector.x;
386     imu_msg.linear_acceleration.y = linear_accel_msg.vector.y;
387     imu_msg.linear_acceleration.z = linear_accel_msg.vector.z;
388     for (int i = 0; i < 9; i++) {
389       imu_msg.orientation_covariance[i] = orientation_covariance[i];
390       imu_msg.angular_velocity_covariance[i] =  ↩
              angular_velocity_covariance[i];
391       imu_msg.linear_acceleration_covariance[i] =  ↩
              linear_acceleration_covariance[i];
392     }
393     imu_pub.publish(&imu_msg);
394     odom_pub.publish(&odom_msg);
395     Serial.println(currentMillis - previousMillis);
396
397     previousMillis = currentMillis;
398
```

```
399
400   }
401
402 }
403 void printEvent(sensors_event_t* event) {
404
405   if (event->type == SENSOR_TYPE_GYROSCOPE) {
406
407     gyro_msg.vector.x = event->gyro.x;
408     gyro_msg.vector.y = event->gyro.y;
409     gyro_msg.vector.z = event->gyro.z;
410   }
411
412   else if (event->type == SENSOR_TYPE_LINEAR_ACCELERATION) {
413
414     linear_accel_msg.vector.x = event->acceleration.x;
415     linear_accel_msg.vector.y = event->acceleration.y;
416     linear_accel_msg.vector.z = event->acceleration.z;
417   }
418 }
```

Listing C.1: Teensy Raspi Interface