University of Turku
Faculty of Technology
Department of Mechanical Engineering

Trajectory Tracking Control of Omnidirectional Robot Using Adaptive Fuzzy Control

Maliha Tabassum

Master's Thesis
Degree Programme in Mechanical Engineering (Smart Systems)

June 26, 2026

Supervised by:
Professor Wallace Moreira Bessa
Dr. Gabriel Da Silva Lima

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

Abstract

Mecanum-wheeled robots can move in any direction and rotate at the same time. This makes them useful in tight industrial and service spaces. But getting them to follow a path accurately is not easy. Wheel–ground friction is nonlinear, the translational and rotational dynamics are coupled and the motor drivers have limited bandwidth. A simple fixed-gain controller can not handle these problems well.

In this thesis is presented an adaptive fuzzy gain-scheduled trajectory tracking controller for the Mecabot2, a four-wheeled mecanum robot running under ROS2. The controller works at the kinematic velocity level sending forward, lateral and angular velocity commands, which are converted to individual wheel speeds using mecanum inverse kinematics. A Mamdani fuzzy inference system replaces the fixed gains of a standard kinematic controller. The gains are computed online from the current tracking error and its rate of change, measured in the robot body frame. Three fuzzy systems run in parallel, one per correction channel. Lyapunov stability analysis shows the tracking errors stay bounded and converge asymptotically to zero.

Gazebo simulation tests on a circular path with correct parameterisation demonstrate accurate trajectory tracking performance, confirming that the controller converges reliably to the reference path. In this thesis, two common failure modes encountered during development: an initialisation error from wrong trajectory centre parameterisation and a phase-lag error from a logging timing mismatch between the odometry subscriber and the controller startup. Both failure modes are diagnosed, explained and corrected. Practical tuning guidelines come out of both findings. Hardware validation on the physical Mecabot2 platform is presented alongside the simulation results to assess real-world performance.

Keywords: mecanum wheel, trajectory tracking, adaptive fuzzy control, Mamdani inference, ROS2, Gazebo, holonomic mobile robot, Lyapunov stability.

Acknowledgements

I am very thankful to Professor Wallace Moreira Bessa for supervising this thesis and for providing access to the Mecabot2 platform and the Smart Systems laboratory. His guidance on the control design and his detailed feedback on the manuscript were essential to the completion of this work. I am also thankful to Dr. Gabriel Da Silva Lima for his advice on the ROS2 implementation, for his help in setting up the Gazebo simulation environment and for the many discussions that shaped the direction of this project.

I would like to thank my colleagues in the robotics laboratory for their willingness to share their experience and for their help in resolving the early implementation difficulties.

This research was supported by Business Finland, under the Veturi programme with the Dining Flow project (6547/31/2022).

Generative AI tools were used during the preparation of this thesis for language editing and grammar.

Turku Maliha Tabassum
June 26, 2026

Contents

List of Figures

List of Tables

List of Abbreviations

AGV	Automated Guided Vehicle
COM	Centre of Mass
DOF	Degree of Freedom
FIS	Fuzzy Inference System
MPC	Model Predictive Control
NB	Negative Big
NS	Negative Small
PB	Positive Big
PID	Proportional-Integral-Derivative
PS	Positive Small
RMSE	Root Mean Square Error
ROS2	Robot Operating System 2
STM32	STMicroelectronics 32-bit microcontroller family
URDF	Unified Robot Description Format
ZO	Zero

Introduction

1.1 Background: Ground Robots and Their Limitations

Most ground robots and transport systems in use today rely on conventional wheels for locomotion. A conventional wheel has one important kinematic restriction: it can only generate traction in the direction of its spin axis. Motion in the perpendicular direction is blocked by friction at the contact patch. As a result a robot that uses conventional wheels must first rotate its heading before it can move in a new direction. Every narrow-aisle task, precision docking operation and tight parking manoeuvre therefore requires a sequence of rotations and forward moves instead of a direct path (Siegwart et al., 2011).

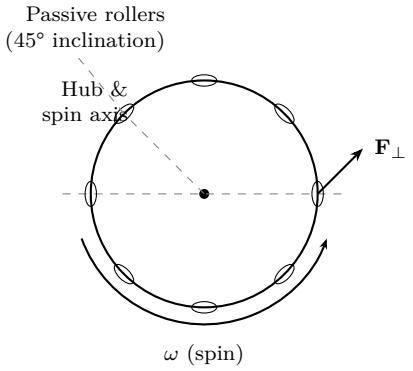
This constraint is acceptable in open environments where turning space is available. In automated warehouses, hospital corridors, aircraft assembly halls and research laboratories the situation is different. Spaces are confined, obstacles appear without warning and the robot must reach precise positions at arbitrary orientations. Under these conditions the inability to move sideways becomes a serious operational limitation (Dixon et al., 2004; Moreno Caireta, 2019).

There are three families of devices that help overcome this limitation associated with omnidirectional driving. Traditional steered (castor) wheels make use of the addition of a motorised steering axis to every wheel but this solution is rather bulky and mechanically complex. Passive Swedish omnidirectional wheels utilise a number of rollers mounted perpendicularly to the wheel rim which allow for omnidirectional motion, though the passive rollers reduce the effective contact area with the ground compared to conventional wheels (Siegwart et al., 2011). Mecanum wheels, first described in a patent filed in 1972 and granted in 1975 by Bengt Erland Ilon of Sweden (Ilon, 1975) provide the optimal compromise. They have the same motorised hub design as ordinary wheels but generate planar omnidirectionality in full with three degrees of freedom using rollers with cylindrical shape tilted at 45 degrees to the spin axis (Muir & Neuman, 1987; Siegwart et al., 2011).

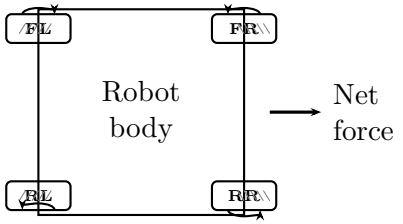
1.2 The Mecanum Wheel: Operating Principle

Each Mecanum wheel consists of a fixed hub carrying a number of passive cylindrical rollers the axes of which are inclined to the wheel spin axis by 45 degrees (Figure 1.1a). In this way, when the hub rotates a roller can only transmit force along the line perpendicular to its axis. Thus, this force has longitudinal and lateral components which can be decomposed in terms of the reference frame attached to the chassis. Then using independent control over the speeds and directions of four wheels installed at the corners of a rectangular chassis it is possible to obtain any desired combination of longitudinal, lateral and rotational movements (Figure 1.1b) (Muir & Neuman, 1987).

It may be noted that this decomposition is linear and is already well researched (Muir & Neuman, 1987; Siegwart et al., 2011). Namely the installation of two left-handed and two right-handed wheels in an X-pattern allows obtaining any linear combination of forces and consequently any type of movement: forward and reverse, lateral and in-place rotation both clockwise and counterclockwise. All these types of motion constitute six elementary modes of motion for Mecanum wheel platforms as illustrated in Figure 1.2.



(a) Single Mecanum wheel
(rollers at 45° to spin axis)



(b) Four-wheel platform (top view)
(example: lateral motion to the right)

Figure 1.1: Mecanum wheel principle of operation. **(a)** An ordinary Mecanum wheel with rollers inclined to the spin axis by 45°. It transmits force perpendicular to its rollers' axis which creates a resultant with both longitudinal and lateral components. **(b)** Top view of a four-wheel configuration designed for lateral motion to the right: diagonally located pairs of wheels are rotated in opposite directions so that lateral forces reinforce each other and longitudinal ones compensate (Muir & Neuman, 1987).

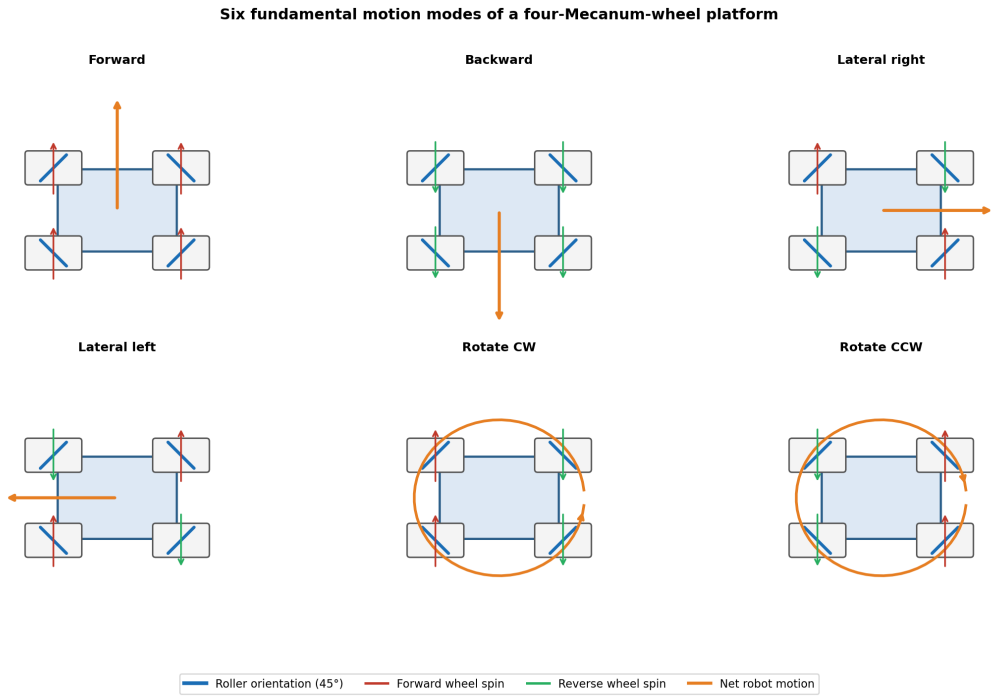


Figure 1.2: Six fundamental motion modes of a four-Mecanum-wheel platform: forward, backward, lateral motion (left/right) and rotation (clockwise and counterclockwise). Source: (Moreno Caireta, 2019).

1.3 Industrial and Service Applications

Due to their freedom of movement and simple design there has been a significant increase in the use of Mecanum-wheeled robots in many different environments. AGVs with Mecanum wheels align with loading docks to millimetre precision. Hospital robots move around beds without disrupting the surrounding space. Research teams use them as test beds for motion planning and formation control (Moreno Caireta, 2019; Siegwart et al., 2011).

In logistics and warehousing Mecanum-wheeled forklifts carry loads into rack positions that are not reachable by conventional trucks without wide turning aisles. In aerospace manufacturing large omnidirectional AGVs transport fuselage sections and wing assemblies along assembly lines where turning space is very limited (Moreno Caireta, 2019). These setups usually require the robot to go from one place to another without colliding with any element of the environment including dynamic obstacles such as human workers or other robots. Some of these applications are shown in Figure 1.3.

1.4 The Mecabot2 Platform and the ROS2 Context

The platform used in this thesis is the Mecabot2, a four-wheeled Mecanum robot developed and maintained in the Smart Systems laboratory at the University of Turku. The

Figure 1.3: Some applications of Mecanum-wheeled platforms. **Top left:** KUKA KMR QUANTEC AGV in aerospace manufacturing. **Top right:** Mecanum-wheeled transporters positioning a train body in an assembly hall. **Bottom:** Airtrax ATX3000 Sidewinder omnidirectional forklift. Source: Moreno Caireta (2019).

Mecabot2 is manufactured by Roboworks and runs on a 24 V 6000 mAh lithium battery pack. One Mecanum wheel is mounted at each corner of a steel chassis each independently driven by a brushed DC gear motor. The onboard electronics stack visible in Figure 1.4 includes an NVIDIA Jetson Orin-based single-board computer running ROS2 Humble an STM32-based motor driver board and supporting power electronics. The motor driver handles low-level torque regulation internally and exposes only a velocity-command interface (`/cmd_vel`) to the high-level ROS2 controller.

The Mecabot2 has four mecanum wheels one at each corner each driven by an independent motor (Figure 1.4). Its STM32 motor driver handles torque internally and only accepts velocity commands from above. The high-level controller has no access to torques or currents (Siegwart et al., 2011).

This rules out backstepping and sliding-mode controllers. Both need direct torque access (Dixon et al., 2004; Ha et al., 2019). The controller in this thesis works at the velocity level instead. It sends body-frame velocity commands that the driver executes. Velocity-level control works well for low-to-medium speed mecanum tracking and the architecture transfers across platforms with the same wheel geometry (Fahmizal & Kuo, 2016; Lima et al., 2022). Model Predictive Control requires solving an optimisation problem at every control cycle; which imposes a significant computational burden and must be completed within the control interval (Moreno Caireta, 2019). On a platform

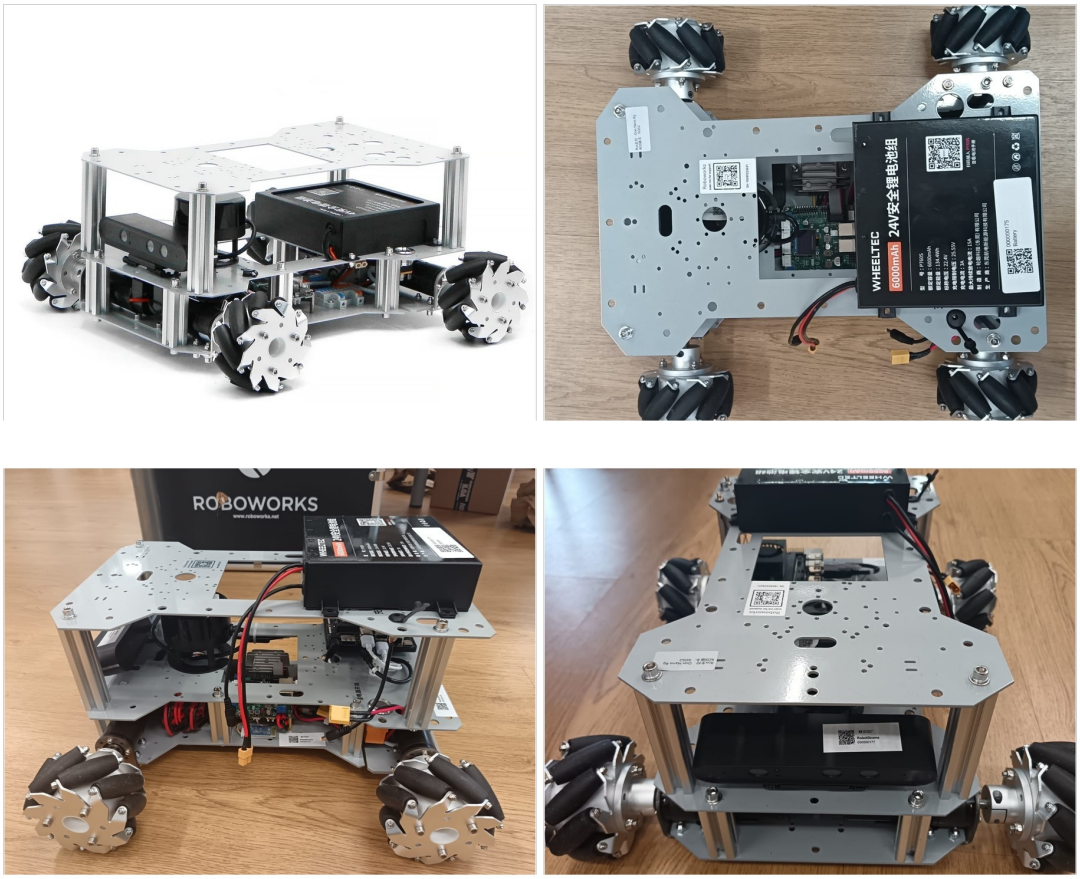


Figure 1.4: The Mecabot2 platform used in this thesis. **Top-left:** Product overview showing the three-tier aluminium chassis and four Mecanum wheels. **Top-right:** Top view showing the 24 V / 6000 mAh WHEELTEC battery pack and the Jetson Orin / STM32 electronics bay. **Bottom-left:** Side view showing internal wiring and motor driver board. **Bottom-right:** Front view showing the Mecanum wheel arrangement and independent motor mounts.

running at 20 Hz and sharing processor resources with other ROS2 processes, this cost may be difficult to justify for the present application.The adaptive fuzzy controller developed in this thesis operates entirely at the velocity level making it directly compatible with the existing Mecabot2 interface and portable to other platforms sharing the same wheel geometry.

ROS2 handles communication, parameters and simulation integration. Every tunable value in the controller can be changed at launch without recompiling. Trajectory geometry, gain limits and velocity bounds are all set through a YAML configuration file. Gazebo provides the simulation environment for validating the design before running on real hardware.

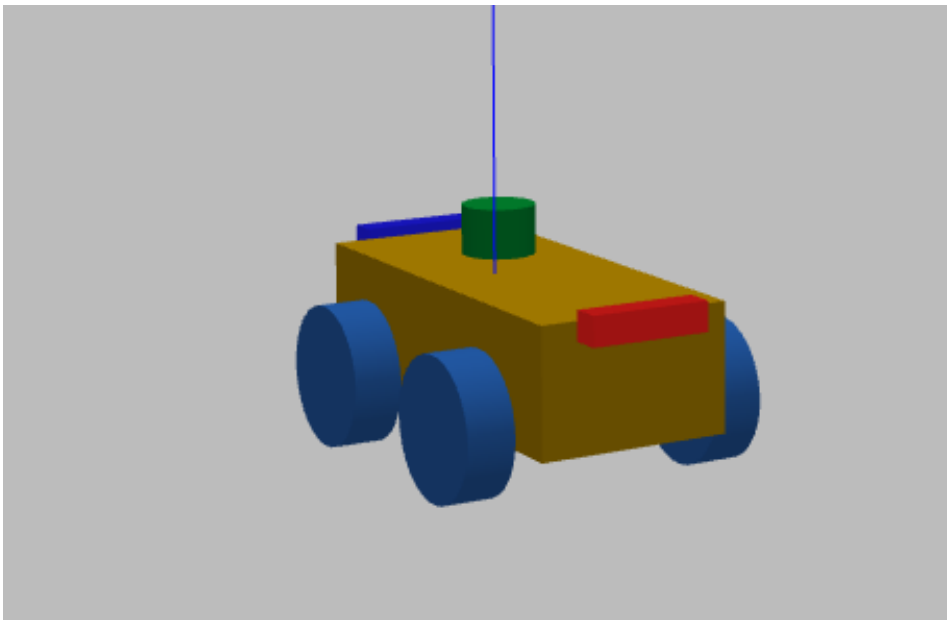


Figure 1.5: The Mecabot2 in Gazebo under ROS2 Humble using the `mecanum.urdf.xacro` description from the linorobot2 package. The robot spawns at the world-frame origin (0, 0) with zero heading for all trajectory tracking experiments.

1.5 Why Trajectory Tracking is Hard

Accurate trajectory tracking on mecanum platforms is not straightforward. The kinematic model assumes ideal rollers. Real rollers deform under load show friction that shifts with heading and floor surface and vibrate as they contact the ground periodically (Moreno Caireta, 2019; Siciliano et al., 2010). Most controllers in the literature ignore these effects.

Fixed-gain controllers have a separate problem. A gain set for smooth circular tracking could respond too slowly to a sudden disturbance. Raise that gain and the robot could overshoot when the error is already small and converging. The right gain depends on the current error and a fixed gain cannot adapt to it.

Fuzzy inference handles this naturally: large error needs large gain and small converging error needs small gain and no explicit disturbance model is required. The output is bounded and runs fast enough for real-time control on embedded hardware (Passino & Yurkovich, 1998; Wang, 1994; Zadeh, 1965). Previous work on wheeled robots confirms it works well in practice (Fahmizal & Kuo, 2016).

1.6 Scope, Objectives and Organisation

In this thesis, a velocity-level adaptive fuzzy gain-scheduled trajectory tracking controller for the Mecabot2 is designed, implemented and evaluated, positioning the work within the broader family of intelligent kinematic controllers for mecanum robots. The specific contribution is a practical ROS2-compatible implementation that combines Mamdani-style

online gain adaptation with a clean Lyapunov stability argument and that is validated in both Gazebo simulation and on the physical hardware. The work involved deriving the body-frame tracking error dynamics, proving baseline controller stability, designing a Mamdani fuzzy gain scheduler with online adaptation, proving stability of the adaptive system via Lyapunov analysis, implementing it in ROS2 and testing it in Gazebo on a circular trajectory. Two failure modes found during development are also documented: a trajectory initialisation error and a phase-lag instability. Both are common in mecanum implementations and both receive practical fixes and tuning guidelines.

Performance is measured using root mean square position error:

$$\mathrm{RMSE}_{xy} = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left[\left(x_d^{(k)} - x^{(k)}\right)^2 + \left(y_d^{(k)} - y^{(k)}\right)^2\right]}, \tag{1.1}$$

computed over the N odometry samples recorded during each experiment.

Chapter 2 reviews the relevant literature. Chapter 3 covers the kinematic model and controller design. Chapter 4 covers ROS2 implementation and Gazebo setup. Chapter 5 presents the simulation and hardware results. Chapter 6 summarises contributions and identifies directions for future work.

Literature Review

This chapter reviews the research background most directly relevant to trajectory tracking control of omnidirectional mobile robots, with particular emphasis on mecanum-wheeled platforms and adaptive fuzzy control. The review is organised around five themes. First, it discusses the modelling and kinematic properties of omnidirectional mobile robots. Second, it examines the main control strategies used for trajectory tracking, including classical kinematic control, robust nonlinear control, model predictive control, neural control and disturbance rejection methods. Third, it focuses on fuzzy logic control and adaptive fuzzy systems, since these form the main methodological basis of this thesis. Fourth, it reviews the software and simulation infrastructure commonly used for robot control development, especially ROS2 and Gazebo. Finally, it identifies the research gaps that motivate the present work and positions this thesis within the current state of the art.

2.1 Omnidirectional Mobile Robots

Wheeled mobile robots are often classified according to the motion constraints imposed by their wheel configuration. Conventional differential-drive and car-like robots are non-holonomic, meaning that they cannot instantaneously generate motion in all planar directions. They must first rotate before translating in a new direction. This restriction is manageable in open spaces but becomes a significant limitation in narrow, cluttered or dynamically changing environments. For this reason omnidirectional mobile robots have received increasing attention in logistics, service robotics, healthcare and industrial transport (Dixon et al., 2004; Siegwart et al., 2011).

Among omnidirectional drive solutions the mecanum wheel remains one of the most practical and widely adopted options. The concept introduced by Ilon (1975) makes it possible to achieve full planar mobility by using passive rollers mounted at an angle around each wheel rim. In a four-wheel X-configuration the platform can produce forward, lateral, diagonal and rotational motion through appropriate wheel speed combinations.

This mechanical simplicity combined with the ability to move sideways without steering explains why mecanum robots have become common in indoor applications where turning space is limited (Moreno Caireta, 2019).

The kinematic modelling of mecanum robots was placed on a rigorous foundation by Muir and Neuman (1987), whose Jacobian-based approach remains central in the field. Their work showed that the mapping between wheel angular velocities and body-frame velocities can be expressed in a compact linear form dependent on wheel radius and chassis dimensions. This linear structure is very important. It means that a high-level controller only needs to specify three body-frame velocity targets forward speed, lateral speed and yaw rate and the inverse kinematics will compute the four individual wheel commands with modest computational cost. On embedded platforms where the low-level driver exposes only velocity inputs this kinematic simplicity is a clear advantage (Ortiz Hernández & Rosas Almeida, 2024). It also means the mapping is invertible, which allows odometry to be derived from wheel encoder readings using the pseudoinverse of the same matrix.

Later work extended this ideal kinematic model to account for more realistic effects such as roller compliance, surface variation, wheel slip and friction anisotropy. Taheri et al. (2015) provided a detailed kinematic model for four-mecanum-wheeled platforms and validated the forward and inverse kinematic equations against experimental results across eight motion directions. While the ideal kinematic model assumes no wheel slip, in practice the contact interaction between the rollers and the floor generates disturbances that are non-negligible even in controlled indoor settings showing up as direction-dependent tracking offsets and occasional velocity variations when a roller transitions between loaded and unloaded contact (Moreno Caireta, 2019; Siciliano et al., 2010). Textbooks such as those by Siegwart et al. (2011) and Siciliano et al. (2010) also note that although mecanum wheels provide full planar mobility in theory, they do not automatically guarantee accurate dynamic behaviour under real contact conditions. The gap between the ideal kinematic model and real hardware performance is therefore an important motivating factor for adaptive control approaches that can compensate online without requiring an explicit model of every disturbance source.

This distinction between ideal kinematics and imperfect real motion is especially important for trajectory tracking. In theory a mecanum robot can follow arbitrary planar velocity commands. In practice roller-ground interaction, chassis vibration, actuator bandwidth, uneven load distribution and wheel slip phenomena introduce model mismatch. The robot may therefore lag behind the reference, overshoot in one direction or show different performance in the longitudinal and lateral channels. This is one reason why purely fixed-gain controllers often perform well only in a narrow operating regime.

The dynamic description of mecanum robots further clarifies this issue. The full planar dynamics include inertial effects, Coriolis-type coupling between translational and

rotational motion and damping effects associated with friction and drag (Rojas & Förster, 2006; Siciliano et al., 2010). The Coriolis coupling is of particular note: it introduces a cross-dependence between the forward and lateral velocity channels through the yaw rate, so that even a purely kinematic controller must account for this coupling in its stability analysis. At low speed many studies justify reducing the dynamic model to a kinematic one for controller design, especially when the hardware interface exposes only velocity inputs. However, even when the kinematic approximation is valid for the nominal motion, unmodelled disturbances remain. These disturbances motivate the use of robust or adaptive control methods that can adjust their corrective action online instead of relying entirely on an accurate plant model (Huang et al., 2025; Wu & Karkoub, 2022).

In summary, the literature on omnidirectional robot modelling establishes three important points for this thesis. First, mecanum robots are especially attractive when lateral mobility is required in confined spaces. Second, their velocity-level kinematic model is sufficiently simple and well understood to support practical controller design. Third, real trajectory tracking performance is strongly affected by unmodelled contact and friction effects, making adaptive feedback strategies particularly relevant.

2.2 Trajectory Tracking Control for Wheeled Mobile Robots

Trajectory tracking is the problem of forcing a robot to follow a desired time-parameterised reference path. Unlike path following, which only requires the robot to converge to a geometric curve, trajectory tracking requires the robot to be at the correct place at the correct time. This makes the problem more demanding because it couples geometric accuracy with timing consistency. For omnidirectional platforms, trajectory tracking typically involves simultaneous control of forward velocity, lateral velocity and yaw rate, all while respecting actuator limits and maintaining stability (Dixon et al., 2004; Siciliano et al., 2010).

The simplest widely used approach is the kinematic proportional controller. In its standard form the tracking error is expressed in the robot body frame and each channel is corrected by a proportional gain. The method is attractive because it is easy to analyse, computationally efficient and directly compatible with velocity-command interfaces. The body-frame error formulation and its Lyapunov stability properties for kinematic controllers have been established in the mobile robotics literature (Dixon et al., 2004; Siegwart et al., 2011). The skew-symmetric structure of the cross-coupling terms in the body-frame error dynamics allows the cross terms to cancel exactly in the Lyapunov derivative yielding a globally asymptotically stable closed-loop system under suitable gain choices. For omnidirectional platforms the body-frame formulation is especially conve-

nient because it aligns the control channels with the robot's actual motion directions and allows each channel to be corrected independently (Ortiz Hernández & Rosas Almeida, 2024; Siegwart et al., 2011).

However, the main limitation of proportional kinematic control is the use of fixed gains. A gain choice that is large enough to correct a major deviation quickly may become too aggressive when the robot is already close to the desired trajectory, producing oscillatory overshoot. Conversely, a gain that gives smooth convergence near the reference may be too weak to reject disturbances or recover from a large transient error. This fixed-gain trade-off appears repeatedly in the literature and motivates gain adaptation (Lima et al., 2022). Several studies have reported that the optimal gain for a mecanum robot varies significantly with trajectory curvature, speed and floor surface, which confirms that a single fixed gain cannot handle the full range of operating conditions encountered in practice.

To overcome the limitations of fixed-gain kinematic control, a wide range of more advanced control methods has been proposed. Model-based nonlinear techniques such as backstepping, dynamic surface control and feedback linearisation seek to compensate for the robot dynamics explicitly. These methods often yield excellent tracking performance in simulation and in well-instrumented experimental platforms. For example, neural-network-assisted backstepping designs have reported very low RMSE values on circular or smooth trajectories (Ha et al., 2019; Mai et al., 2021). However, many of these methods require access to torque or current-level actuation or assume that the low-level control loop can be modified. That assumption is not valid for all practical robots .

Sliding-mode control is another prominent family of methods. It is attractive because of its robustness to bounded uncertainty and disturbances. The approach works by forcing the system onto a sliding surface and keeping it there, so that disturbances within a bounded set are rejected regardless of their source. Recent studies have applied sliding-mode ideas to omnidirectional robots to improve robustness under uncertain friction and external perturbation. For example, Wu and Karkoub (2022) addressed friction compensation through cascaded sliding-mode control for an uncertain omnidirectional mobile robot, demonstrating that systematic friction models can be incorporated into the control law to reduce steady-state tracking error. More recent hybrid designs also combine fuzzy logic with sliding-mode control to improve tracking robustness while reducing some of the well-known disadvantages of pure sliding-mode approaches such as chattering (Chen, 2025). These results confirm that high tracking accuracy is achievable, but they also highlight the usual trade-offs: the need to bound disturbances explicitly, higher implementation complexity and stronger dependence on plant model assumptions (Khalil, 2002).

Model Predictive Control (MPC) has also become an important direction in mobile robot trajectory tracking research. MPC is attractive because it can explicitly handle

constraints on velocity, acceleration and actuator saturation while optimising tracking behaviour over a finite prediction horizon. By solving a constrained optimisation problem at each control step, MPC can anticipate future trajectory deviations and pre-compensate for them, which is particularly beneficial for aggressively curved paths or paths with abrupt direction changes. Moreno Caireta (2019) studied MPC for a mecanum-wheeled robot in dynamic environments and demonstrated that the controller can handle velocity limits and obstacle avoidance constraints in real time. More recent work has extended this direction to adaptive and trajectory-smoothing contexts. Li et al. (2023) proposed an adaptive MPC strategy for trajectory tracking of mecanum mobile robots, while Carvalho et al. (2024) used B-spline trajectory representation within an MPC framework for omnidirectional robots, allowing the trajectory itself to be optimised alongside the control input. These studies show that MPC is highly capable in structured environments, especially when constraint handling is important. Yet the computational cost of solving a quadratic programme at every control step remains a concern for embedded systems with modest control frequencies. On a platform where the outer loop runs around 20 Hz and shares processor resources with other ROS2 processes, this complexity may not be justified unless the application demands constraint-rich optimal planning.

Active disturbance rejection and related robust designs also appear in the recent literature. These methods are motivated by the same practical issue discussed earlier: real mecanum motion is strongly affected by unmodelled disturbances and waiting for those disturbances to produce a large tracking error before reacting is inefficient. By estimating disturbances online and compensating for them in the control input before they grow large, such methods can significantly improve tracking performance. For example, Lamraoui and Nemra (2024) proposed an active disturbance rejection approach specifically for four-mecanum-wheel robots and demonstrated significant improvements in tracking under uncertain load and surface conditions. Likewise, Huang et al. (2025) combined robust trajectory tracking with obstacle avoidance for omnidirectional robots operating in unstructured environments. These methods share the advantage of better robustness to uncertain friction, varying load and external disturbances. Their disadvantage is increased controller complexity and a heavier tuning burden, since the disturbance observer itself requires careful design and the combined system has many more parameters than a simple proportional controller.

Neural network-based control has similarly gained attention in the trajectory tracking literature. In this framework neural adaptive control is presented as a way to approximate unknown inverse dynamics or nonlinear disturbance structure using a function approximator that updates its weights online. Lima et al. (2022) showed that adaptive neural control can produce accurate trajectory tracking for omnidirectional robots while providing a Lyapunov-based stability argument for the weight update law. This work is particularly relevant because it operates on a related class of omnidirectional robot and

uses a similar stability framework, confirming that Lyapunov analysis is both applicable and informative for this class of controllers. Feng and Wang (2022) also demonstrated adaptive neural network tracking for an omnidirectional mobile robot using online weight adaptation, reporting good robustness under varying conditions. Other work such as Zhao et al. (2023) combines self-organising fuzzy neural networks with preview strategies to improve tracking on complex paths by exploiting look-ahead information about the upcoming trajectory. These methods are powerful, but they are often less transparent than rule-based fuzzy systems because the learned weights do not correspond directly to interpretable physical quantities. They may also require training procedures or architectural choices that are difficult to justify in a practical engineering thesis focused on deployment and interpretability.

For mecanum robots specifically, the literature shows a clear progression from classical PID and fixed-gain kinematic controllers toward more robust, intelligent and adaptive designs. Fahmizal and Kuo (2016) demonstrated that a Mamdani fuzzy controller combined with IMU-based heading feedback can improve circular trajectory tracking over pure kinematic control on a mecanum robot, providing an early benchmark for fuzzy approaches on this platform type. More recent papers now include adaptive MPC, robust trajectory tracking, fuzzy-neural control and hybrid fuzzy-sliding-mode tracking (Chen, 2025; Huang et al., 2025; Li et al., 2023; Zhao et al., 2023). This progression shows that adaptive and intelligent control for mecanum and omnidirectional robots is a well-established and active research area. However, relatively few papers focus specifically on the practical deployment scenario where the controller must operate purely at the velocity-command level because the low-level driver does not expose torque access and where interpretability and ROS2 compatibility are primary design constraints.

2.3 Fuzzy Logic Control and Adaptive Fuzzy Systems

Fuzzy logic control originates from the work of Zadeh (1965) who introduced fuzzy sets as a mathematical framework for representing partial membership and linguistic uncertainty. The core idea is that real-world concepts such as large, small, fast or slow do not have sharp boundaries. Rather than requiring a precise threshold to distinguish large from small, a fuzzy set assigns a membership degree between zero and one to every value, allowing smooth transitions between linguistic categories. This represents human knowledge and engineering intuition more faithfully than crisp Boolean logic, because experts naturally reason about control problems using imprecise but practically useful rules such as if the error is large then apply a strong correction.

Mamdani and Assilian (1975) demonstrated that this idea could be applied directly in control by encoding expert knowledge into if-then rules and evaluating them using fuzzy inference. The resulting Mamdani controller proved capable of controlling a steam engine

without an explicit mathematical model of the plant which established fuzzy inference as a viable control design methodology. Subsequent theoretical work showed that fuzzy systems are universal approximators capable of representing any continuous function on a compact domain to arbitrary accuracy with a sufficient number of rules (Wang, 1994). This universality property provides theoretical justification for applying fuzzy systems to complex nonlinear control problems even when the plant model is incomplete or unknown.

This feature makes fuzzy control especially suitable for mobile robots. In many robot control problems the corrective logic is intuitive even when the disturbance mechanisms are not fully characterised. For example, one may not know the exact friction coefficient or slip model at every instant but it is reasonable to say that if the position error is large the controller should respond more strongly and if the error is small and already decreasing the response should be milder. The rate of change of the error adds further resolution: a large error that is growing requires more urgent action than a large error that is already converging rapidly. Fuzzy controllers translate this two-dimensional intuition into a structured inference mechanism that produces bounded, smooth control outputs (Passino & Yurkovich, 1998).

The literature distinguishes between fixed fuzzy controllers and adaptive fuzzy controllers. In a fixed fuzzy controller, the membership functions, rule base and gain levels are designed offline and remain unchanged during operation. Fahmizal and Kuo (2016) applied this approach to mecanum trajectory tracking and showed that Mamdani inference can improve tracking quality relative to pure kinematic control. The improvement comes from the nonlinear mapping between error and corrective action that the fuzzy rules implement: rather than a fixed proportional gain, the effective gain varies continuously with the error magnitude and rate. However, the membership functions and output singletons in that system are fixed after design-time tuning which means the controller still inherits some of the same sensitivity to operating condition changes as a classical fixed-gain method even though the error-to-output mapping is more expressive.

Adaptive fuzzy control goes further by allowing some part of the fuzzy controller to change online during operation. Different papers achieve this in different ways. Some approaches adapt the rule consequents using gradient descent or recursive least squares. Others adapt the membership function centres and widths to better fit the observed plant behaviour. A third approach uses the fuzzy system as a function approximator for unknown dynamics and updates the approximation parameters using a Lyapunov-based adaptation law. Wang (1994) provided the major theoretical foundation for this last approach by linking fuzzy systems to adaptive control theory and showing that the approximation error can be driven to zero asymptotically under appropriate conditions. This framework made it possible to design adaptation laws with formal stability guarantees rather than relying on heuristic tuning (Nguyen Minh et al., 2023).

Recent literature shows several active branches of adaptive fuzzy control for mobile

robots. One branch combines fuzzy adaptation with more classical control structures to produce designs that are relatively easy to understand and deploy. For example, Mai et al. (2021) combined backstepping with an adaptive fuzzy PID approach and demonstrated stable trajectory tracking for autonomous mobile robots. The fuzzy component adapts the PID gains online based on the observed error while the backstepping structure provides a systematic way to handle the robot's nonlinear dynamics. Similarly, Cao et al. (2022) applied fuzzy adaptive PID to multi-mecanum-wheeled robots where the PID gains are adjusted online through fuzzy rules. The advantage of this class of methods is that they remain close to familiar PID practice while adding online adaptability.

Another branch combines fuzzy logic with neural or robust nonlinear methods to obtain stronger approximation capability and broader robustness. For instance, Zhao et al. (2023) proposed a self-organising fuzzy neural network approach for omnidirectional robot tracking where the network structure itself evolves during operation to accommodate new operating conditions. The self-organisation allows the approximator to allocate additional rules automatically when it encounters regions of the error space where its current representation is inadequate. While this gives the approach great flexibility, it also makes the controller harder to analyse and interpret. Similarly, Chen (2025) developed a nonlinear adaptive fuzzy hybrid sliding-mode controller for trajectory tracking of autonomous mobile robots. The sliding-mode component provides robustness to bounded disturbances while the fuzzy component softens the chattering that pure sliding-mode produces. These hybrid designs aim to capture the strengths of both frameworks but the resulting controllers are considerably more complex than pure fuzzy gain schedulers.

A further line of work focuses on adaptive fuzzy tracking with explicit performance guarantees. For example, Ding et al. (2024) proposed an adaptive fuzzy controller with prescribed trajectory tracking performance for wheeled mobile robots ensuring that the tracking error remains within a predefined bound that decreases according to a specified time profile. This prescribed performance framework is theoretically appealing because it allows the designer to specify acceptable error bounds a priori rather than relying on asymptotic convergence alone. However, the implementation typically requires barrier functions and error transformation techniques that add considerable complexity to the control architecture.

For the present thesis, the most relevant fuzzy control design choice is between Mamdani and Takagi–Sugeno–Kang (TSK) inference. TSK systems use crisp polynomial functions as rule consequents which makes them computationally efficient and convenient for certain theoretical developments. Mamdani systems, by contrast, use fuzzy sets as consequents which makes the rule base directly readable and editable by a human designer. Each rule can be interpreted as a natural language statement such as if the forward error is positive big and the error rate is negative small, then the forward gain should be medium. This interpretability is valuable in a practical engineering setting because it allows rules

to be tuned based on physical reasoning rather than purely numerical optimisation. Since the goal of this thesis is not only tracking performance but also a practical and understandable design that can be inspected and modified by future researchers, the Mamdani framework is well justified (Fahmizal & Kuo, 2016; Passino & Yurkovich, 1998).

The reviewed literature also supports this choice indirectly. Many recent adaptive fuzzy papers that combine fuzzy logic with observers, neural approximators or hybrid robust control structures achieve high tracking accuracy but at the cost of considerably increased complexity (Chen, 2025; Ding et al., 2024; Zhao et al., 2023). A Mamdani-style gain scheduler occupies a different design point: the fuzzy system adjusts the correction gain online based on the current error and its rate of change without attempting to approximate the full robot dynamics. This keeps the controller lean, fast and interpretable while still providing the online adaptability that fixed-gain methods lack. For the Mecabot2, whose hardware interface is velocity-level, this simplicity is a practical advantage rather than a limitation.

2.4 Simulation and Software Infrastructure

The software and simulation environment deserves attention because controller performance in robotics is shaped not only by the algorithm but also by the implementation pipeline. ROS2 has become the standard middleware for robot system integration, especially in modern research and industrial prototyping (Lima et al., 2022). Its publish-subscribe communication model allows software components to exchange typed messages over named topics without direct coupling between nodes. This architecture means that the trajectory generator, the controller node, the odometry estimator and the simulation environment can all be developed and tested independently and that the same controller binary can be connected to a simulated robot in Gazebo or a real robot with no code changes as long as the topic names and message types match. For adaptive control development this decoupling is especially valuable because it allows individual components to be replaced or reconfigured without affecting the rest of the system.

One of ROS2's most useful features for controller tuning is the parameter server which allows gains, trajectory settings and controller bounds to be changed at launch time through a YAML file or command-line argument without recompiling. This capability is particularly important for fuzzy and adaptive control where practical tuning typically requires many repeated experiments with different gain ranges, normalisation scales and trajectory speeds. Exposing all tunable quantities through ROS2 parameters makes the development cycle faster and more reproducible. It also improves transparency because the final thesis can report the exact configuration used in each experiment, allowing others to reproduce the results on the same hardware.

Gazebo complements ROS2 by providing rigid-body simulation with contact mod-

elling and topic-level integration with the same interfaces that are later used on hardware (Siegwart et al., 2011). The Gazebo mecanum drive plugin models each wheel as a single contact point with anisotropic friction coefficients which captures the main directional asymmetry of mecanum rollers while remaining computationally tractable. This makes it possible to test a controller in a realistic closed-loop environment before running it on the real robot. Nguyen Minh et al. (2023) validated a fuzzy adaptive controller for a mecanum robot using Gazebo and ROS demonstrating closed-loop trajectory tracking in simulation. At the same time, the literature warns that simulation contact models are simplified relative to real mecanum rollers, so simulation results should be treated as a validation stage rather than a final substitute for hardware experiments. In particular, the single-contact-point model cannot capture roller vibration or speed- dependent slip which are important contributors to tracking error on real hardware.

Recent trajectory tracking papers increasingly emphasise the importance of complete development workflows covering controller design, simulation validation, parameter tuning and transfer to the physical robot. Yet many papers still focus mainly on the mathematical control law and provide limited detail about software architecture, topic interfaces, parameterisation, logging or common failure modes. This is one practical area where a ROS2-centred thesis can make a useful contribution. A controller that is mathematically correct but difficult to reproduce or deploy has limited practical value. By contrast, a fully parameterised ROS2 implementation that moves from Gazebo to a real robot through the same `/cmd_vel` and `/odom` interfaces without code modification is highly useful for future work on the same platform.

2.5 Research Gaps and Positioning of this Thesis

The reviewed literature shows that trajectory tracking of omnidirectional and mecanum robots is already a mature and active research area but specific practical challenges remain once the problem is viewed from the perspective of a ROS2 mecanum platform with a velocity-only hardware interface and a requirement for interpretable, easily deployable control.

The first consideration is design space. Recent papers clearly include neural, fuzzy-neural, adaptive fuzzy, robust and MPC-based approaches, all achieving good results (Chen, 2025; Huang et al., 2025; Li et al., 2023; Zhao et al., 2023). However, many of these methods rely on torque-level modelling, heavier optimisation loops or complex hybrid designs that are not directly deployable on platforms where only velocity commands are accessible. The present thesis focuses on a practical design space where the controller must operate at the velocity-command level using interpretable Mamdani-style fuzzy inference for online gain adaptation within a standard ROS2 `/cmd_vel` architecture. This design point is not unique in the broader literature but its specific combination of velocity-level,

Lyapunov-stable, Mamdani gain scheduling, ROS2-native which has practical value for researchers working on similarly constrained platforms.

The second consideration is interpretability and tuning transparency. Neural and hybrid intelligent controllers can be powerful but they often make it difficult to explain exactly why the controller changes its behaviour at a particular operating point and they typically provide fewer handles for manual adjustment (Feng & Wang, 2022; Zhao et al., 2023). A fuzzy gain scheduler expresses its adaptation logic in directly readable rules such as large error with growing rate needs maximum gain making the controller transparent and easy to reason about. For a platform like the Mecabot2 used in low-speed indoor tasks, this interpretability supports both deployment and future modification.

The third consideration is documentation of development-time failure modes. Much of the literature reports final RMSE values for controllers that have already been tuned and debugged, but provides limited detail about how incorrect trajectory parameterisation, phase lag or bandwidth mismatch were diagnosed and corrected in a real implementation. Such failure modes are common in practice and often consume substantial development time. The present thesis documents two failure modes encountered during the development of this controller ; a trajectory initialisation error caused by incorrect centre offset parameterisation and a phase-lag error caused by an odometry logging timing mismatch between the subscriber and the controller startup together with their diagnostic signatures and practical fixes. Documenting these experiences contributes something useful to future researchers beyond the control algorithm itself.

The fourth consideration is workflow integration. The present work provides not only a controller design but also a complete software pipeline covering the ROS2 node implementation, Gazebo simulation setup, odometry logging, CSV- based data export and post-processing scripts for generating the result plots. This reproducible workflow lowers the barrier for future researchers to extend or compare against this work on the Mecabot2 platform.

System Modelling and Controller Design

Building the controller starts with a mathematical description of how the robot moves. This chapter goes through that in order: coordinate frames and platform geometry, kinematic relationships between wheel speeds and robot motion, tracking error formulation and the step-by-step design of the adaptive fuzzy controller with a stability proof.

3.1 Kinematic and Dynamic Modelling

Two coordinate frames describe the robot's position and orientation in the plane, following standard convention (Siciliano et al., 2010; Siegwart et al., 2011). The inertial frame $\mathcal{F}_I$ is fixed to the ground. The body frame $\mathcal{F}_B$ is attached to the geometric centre of the chassis and moves with the robot. The robot pose at any instant is

$$q(t) = \begin{bmatrix} x(t) \\ y(t) \\ \phi(t) \end{bmatrix}, \tag{3.1}$$

where x and y are Cartesian coordinates in $\mathcal{F}_I$ and ϕ is the heading angle measured counter-clockwise from the positive x_I-axis. The body-frame velocity is

$$\nu(t) = \begin{bmatrix} V_x \\ V_y \\ \omega \end{bmatrix}, \tag{3.2}$$

where V_x is forward velocity, V_y is lateral velocity and $\omega = \dot{\phi}$ is the yaw rate. Transforming body-frame velocity to global pose rate uses the rotation matrix

$$R(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.3}$$

giving $\dot{q} = R(\phi)\,\nu$ (Siegwart et al., 2011) or in scalar form:

$$\dot{x} = V_x \cos\phi - V_y \sin\phi, \tag{3.4}$$

$$\dot{y} = V_x \sin\phi + V_y \cos\phi, \tag{3.5}$$

$$\dot{\phi} = \omega. \tag{3.6}$$

For the mecanum platform the four wheel angular velocities $\omega_w = [\omega_1\ \omega_2\ \omega_3\ \omega_4]^T$ (FL, FR, RL, RR) relate to body-frame velocity through the inverse kinematic mapping (Muir & Neuman, 1987):

$$\omega_w = \frac{1}{r} \begin{bmatrix} 1 & -1 & -L \\ 1 & 1 & L \\ 1 & 1 & -L \\ 1 & -1 & L \end{bmatrix} \nu, \tag{3.7}$$

where r is the wheel radius and $L = \ell + w$ is the sum of the longitudinal and lateral chassis half-dimensions. Each row encodes the contribution of forward speed, lateral speed and yaw rate to one wheel. The negative lateral coefficient of the FL wheel arises from its roller inclination angle of -45ř to the spin axis. As shown by Muir and Neuman (1987), each roller can only transmit force perpendicular to its own axis, so when the robot moves laterally, the -45ř inclination causes the resulting force component to oppose the wheel spin direction rather than reinforce it, hence the negative sign in the first row of (3.7).

$$\nu = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{L} & \frac{1}{L} & -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \omega_w. \tag{3.8}$$

For theoretical completeness the planar dynamics are derived using Euler–Lagrange (Khalil, 2002; Siciliano et al., 2010). The robot moves on a flat plane so gravitational potential energy is constant and drops out. The complete dynamic model is (Siciliano et al., 2010):

$$M\dot{\nu} + C(\nu)\,\nu + D\,\nu = \tau. \tag{3.9}$$

The three terms on the left-hand side each represent a distinct physical contribution. The

first term $M\dot{\nu}$ is the inertial resistance to acceleration. The kinetic energy is

$$T = \tfrac{1}{2}m\Big(V_x^2 + V_y^2\Big) + \tfrac{1}{2}I_z\,\omega^2 = \tfrac{1}{2}\nu^T M\nu, \tag{3.10}$$

where the inertia matrix is

$$M = \mathrm{diag}(m,\, m,\, I_z). \tag{3.11}$$

M is diagonal because the body frame is aligned with the principal axes of the chassis. The three channels have independent inertial contributions and no off-diagonal coupling.

The second term $C(\nu)\nu$ arises from the rotating body frame. Even at constant body-frame speed the velocity direction changes in the inertial frame which appears as an apparent force. This Coriolis effect gives (Siciliano et al., 2010):

$$C(\nu) = \begin{bmatrix} 0 & -m\omega & 0 \\ m\omega & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{3.12}$$

The off-diagonal entries $\pm m\omega$ couple the longitudinal and lateral channels through the yaw rate. The yaw dynamics have no Coriolis coupling in the planar case so the third row and column of C are zero.

The third term $D\nu$ accounts for energy dissipated through roller-ground friction and motor drag, modelled as viscous damping (Siciliano et al., 2010):

$$D = \mathrm{diag}(d_x,\, d_y,\, d_\phi), \tag{3.13}$$

where d_x, d_y and d_ϕ are positive damping coefficients from deceleration experiments.

In practice the Mecabot2's STM32 driver accepts velocity commands and handles torque regulation internally. The high-level controller only produces body-frame velocity commands ν^* (Fahmizal & Kuo, 2016; Lima et al., 2022). At the speeds used here, below 0.3 m/s translation and below 1.0 rad/s yaw, the Coriolis term is small. Its largest entry is $m|\omega||V_y| \le 3.5 \times 1.0 \times 0.3 = 1.05\,\mathrm{N}$, comparable to viscous damping but small next to the maximum wheel driving force of about 8 N per wheel. The inertial term $M\dot{\nu}$ is also small during the smooth trajectory segments where peak accelerations stay below $0.1\,\mathrm{m/s^2}$.

The Coriolis force becomes comparable to the maximum driving force when $m|\omega||V_y| \approx 8\,\mathrm{N}$, which for $m = 3.5\,\mathrm{kg}$ requires $|\omega||V_y| \approx 2.3\,\mathrm{m/s{\cdot}rad/s}$. At the operating speeds here ($|\omega| \le 1.0\,\mathrm{rad/s}$, $|V_y| \le 0.3\,\mathrm{m/s}$) the product is 0.3, about eight times below that threshold. The approximation breaks down above roughly 0.8 m/s translation or 2.5 rad/s yaw. Under the low-speed conditions of this thesis the dynamic model (3.9) reduces to the kinematic model (3.4)–(3.6) for the outer control loop.

3.2 Tracking Error and Controller Design

Let $q_d(t) = [x_d, y_d, \phi_d]^T$ be the desired pose trajectory and $\nu_d(t) = [V_{xd}, V_{yd}, \omega_d]^T$ the corresponding desired body-frame velocity. The deviation is best expressed in the robot body frame because the three channels are kinematically decoupled and can be controlled independently. Rotating the global position error by $-\phi$ gives the body-frame tracking error (Dixon et al., 2004; Siegwart et al., 2011):

$$\begin{bmatrix} E_x \\ E_y \\ E_\phi \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_d - x \\ y_d - y \\ \phi_d - \phi \end{bmatrix}. \tag{3.14}$$

Here E_x is error along the forward direction, E_y along the lateral direction and E_ϕ is heading error. Differentiating (3.14) and substituting the kinematic model (3.4)–(3.6) gives the nonlinear error dynamics (Dixon et al., 2004; Siegwart et al., 2011):

$$\dot{E}_x = V_{xd}\cos E_\phi - V_x + \omega E_y, \tag{3.15}$$

$$\dot{E}_y = V_{xd}\sin E_\phi - V_y - \omega E_x, \tag{3.16}$$

$$\dot{E}_\phi = \omega_d - \omega. \tag{3.17}$$

The heading error channel (3.17) depends only on the yaw rate error and is decoupled from the position errors. The position channels each have a feedforward term, a direct velocity control term and a cross-coupling term through ω. The coupling terms $+\omega E_y$ and $-\omega E_x$ have opposite signs. That skew-symmetric structure is important for the stability proof.

The baseline control law cancels the nonlinear feedforward terms and adds proportional feedback (Dixon et al., 2004; Siegwart et al., 2011):

$$V_x = V_{xd}\cos E_\phi + k_x E_x, \tag{3.18}$$

$$V_y = V_{xd}\sin E_\phi + k_y E_y, \tag{3.19}$$

$$\omega = \omega_d + k_\phi E_\phi, \tag{3.20}$$

where $k_x, k_y, k_\phi > 0$. Substituting into (3.15)–(3.17) gives the closed-loop error dynamics:

$$\dot{E}_x = -k_x E_x + \omega E_y, \tag{3.21}$$

$$\dot{E}_y = -k_y E_y - \omega E_x, \tag{3.22}$$

$$\dot{E}_\phi = -k_\phi E_\phi. \tag{3.23}$$

Stability follows from the Lyapunov function candidate

$$V = \tfrac{1}{2}\left(E_x^2 + E_y^2 + E_\phi^2\right), \tag{3.24}$$

which is positive definite and zero only when all errors vanish. Its time derivative along system trajectories is

$$\begin{aligned}\dot{V} &= E_x\dot{E}_x + E_y\dot{E}_y + E_\phi\dot{E}_\phi \\ &= -k_x E_x^2 + \omega E_x E_y - k_y E_y^2 - \omega E_y E_x - k_\phi E_\phi^2.\end{aligned} \tag{3.25}$$

The cross-coupling terms $+\omega E_x E_y$ and $-\omega E_y E_x$ cancel exactly regardless of ω. The derivative reduces to

$$\dot{V} = -k_x E_x^2 - k_y E_y^2 - k_\phi E_\phi^2 \leq 0. \tag{3.26}$$

Since $\dot{V}$ is negative definite, $E = 0$ is globally asymptotically stable (Khalil, 2002). Tracking errors converge to zero from any initial condition.

3.3 Adaptive Fuzzy Gain Scheduler

The baseline controller is stable but its fixed gains are a practical problem. A gain tuned for large errors overshoots when the error is small. A gain tuned for near-zero tracking responds too slowly to sudden deviations. What is needed is a gain that is large when the error is large and small when it is converging. A fuzzy inference system is well suited to encoding exactly this kind of rule, since it naturally maps linguistic descriptions of error magnitude and rate of change to bounded, smooth control outputs.

The adaptive fuzzy controller replaces k_x, k_y, k_ϕ with online gains from three independent Mamdani fuzzy inference systems (Fahmizal & Kuo, 2016; Mamdani & Assilian, 1975; Nguyen Minh et al., 2023). The Mamdani structure was chosen because its rule consequents are fuzzy sets rather than crisp functions which makes each rule directly readable as a linguistic statement and allows the gain-scheduling logic to be inspected and adjusted without rewriting the underlying mathematics.

$$V_x = V_{xd}\cos E_\phi + K_x(E_x, \dot{E}_x)\, E_x, \tag{3.27}$$

$$V_y = V_{xd}\sin E_\phi + K_y(E_y, \dot{E}_y)\, E_y, \tag{3.28}$$

$$\omega = \omega_d + K_\phi(E_\phi, \dot{E}_\phi)\, E_\phi. \tag{3.29}$$

Each fuzzy system takes the current error e_i and its rate of change $\dot{e}_i$ as inputs and returns the gain K_i. Although the three channels operate on physically different quantities (position errors in metres and heading error in radians), the same membership function partition applies to all three. Before inference, each error and error-rate signal is divided

by a channel-specific normalisation scale, reducing it to a dimensionless value. The normalisation scales are 0.12 m and 0.12 m/s for the forward channel, 0.10 m and 0.10 m/s for the lateral channel and 0.12 rad and 0.12 rad/s for the heading channel. After this step all three normalised inputs lie in the same dimensionless range, so a single set of triangular membership functions can be used uniformly across channels. The choice of scale for each channel determines what error magnitude the fuzzy system treats as "large": a forward error of 0.12 m and a heading error of 0.12 rad are both mapped to the boundary of the normalised universe and receive the same linguistic label. Using the derivative lets the controller tell apart a large error that is growing from one that is already decreasing fast. Each input variable has five triangular membership functions: Negative Big (NB), Negative Small (NS), Zero (ZO), Positive Small (PS) and Positive Big (PB) distributed symmetrically over the normalised input range. A triangular function with centre c and feet a, b evaluates to (Passino & Yurkovich, 1998; Wang, 1994):

$$\mu(x) = \max\left(0,\, \min\left(\frac{x-a}{c-a},\, \frac{b-x}{b-c}\right)\right), \tag{3.30}$$

returning zero outside $[a, b]$ and rising linearly to one at c. Adjacent functions overlap by 50%, so at most four rules fire for any input pair.

The rule base has $5 \times 5 = 25$ rules per channel. Large errors get large gains; small or converging errors get small gains. The rule table is shown in Table 3.1.

Table 3.1: Fuzzy rule base for the gain scheduler. Rows: error e_i; Columns: error rate $\dot{e}_i$; Entries: output gain level where $\mathrm{H} = K_{\max}$, $\mathrm{M} = K_{\mathrm{mid}}$, $\mathrm{L} = K_{\min}$. The same table is used for all three channels.

$e_i \backslash \dot{e}_i$	NB	NS	ZO	PS	PB
NB	H	H	H	M	M
NS	H	M	M	M	L
ZO	M	M	L	M	M
PS	L	M	M	M	H
PB	M	M	H	H	H

For input pair $(e_i, \dot{e}_i)$ the firing strength of rule (j, k) is $\alpha_{jk} = \min(\mu_{A_j}(e_i),\, \mu_{B_k}(\dot{e}_i))$. Fired rules are aggregated by pointwise maximum. The crisp output gain comes from centroid defuzzification (Passino & Yurkovich, 1998). The defuzzification step converts the aggregated fuzzy output μ_{agg} back into a single crisp gain value K_i by computing the centre of mass of the aggregated membership function over the output universe $\Omega = [K_{\min}, K_{\max}]$. Intuitively, if many strong rules fire toward a large gain region, the centre of mass shifts upward and the output gain increases accordingly. If the error is small and converging, fewer rules fire and the centre of mass shifts toward the lower end of Ω producing a smaller

corrective gain:

$$K_i = \frac{\int_\Omega z\, \mu_{\text{agg}}(z)\, \mathrm{d}z}{\int_\Omega \mu_{\text{agg}}(z)\, \mathrm{d}z}, \tag{3.31}$$

Figure 3.1 shows the block diagram of the adaptive fuzzy trajectory tracking controller. The reference trajectory generator supplies the desired pose $\mathbf{q}_d(t)$ and velocity $\boldsymbol{\nu}_d(t)$. The body-frame error block computes $\mathbf{E}=[E_x, E_y, E_\phi]^\mathsf{T}$ from odometry feedback. Three parallel Mamdani fuzzy inference systems produce the adaptive gains K_x, K_y, K_ϕ which the control law combines with the feedforward terms to generate the body-frame velocity command $\boldsymbol{\nu}^*$.

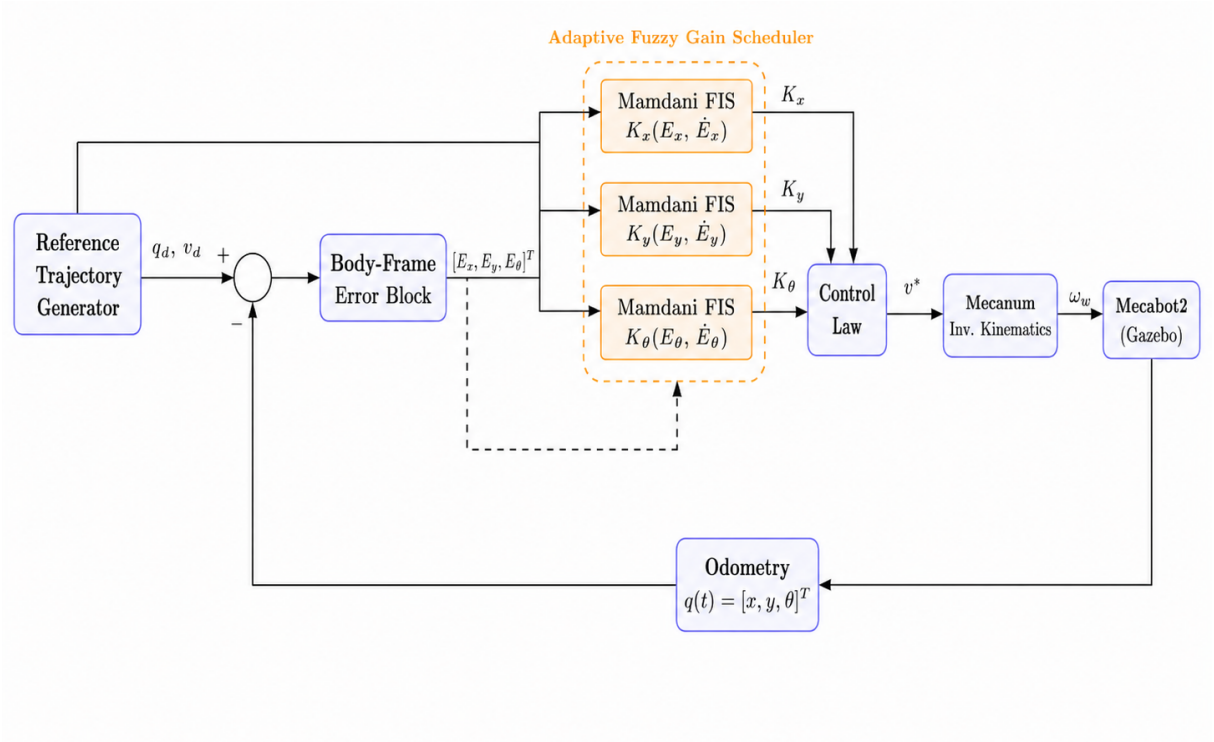


Figure 3.1: Adaptive fuzzy trajectory tracking controller architecture.

The controller is termed adaptive because the gains K_x, K_y and K_ϕ are not fixed at design time but recomputed at every control cycle from the current tracking error and its rate of change. This distinguishes it from a fixed fuzzy controller where the same gain value would be applied regardless of the instantaneous error level.

Stability of the Adaptive Fuzzy System

The closed-loop error dynamics under (3.27)–(3.29) are:

$$\dot{E}_x = -K_x(E_x, \dot{E}_x)\, E_x + \omega E_y, \tag{3.32}$$

$$\dot{E}_y = -K_y(E_y, \dot{E}_y)\, E_y - \omega E_x, \tag{3.33}$$

$$\dot{E}_\phi = -K_\phi(E_\phi, \dot{E}_\phi)\, E_\phi. \tag{3.34}$$

Because the fuzzy output universe is $[K_{\min}, K_{\max}]$ with $K_{\min} > 0$, the gains are bounded and strictly positive by construction (Wang, 1994):

$$0 < K_{\min} \le K_i(e_i, \dot{e}_i) \le K_{\max} \quad \forall\, (e_i, \dot{e}_i). \tag{3.35}$$

Using the same Lyapunov function (3.24), after the same cross-term cancellation:

$$\dot{V} = -K_x E_x^2 - K_y E_y^2 - K_\phi E_\phi^2 \le -K_{\min}\Big(E_x^2 + E_y^2 + E_\phi^2\Big) = -2K_{\min}\, V. \tag{3.36}$$

The inequality $\dot{V} \le -2K_{\min}V$ has the form of the scalar differential inequality $D^+v \le f(t,v)$ with $f(t,v) = -2K_{\min}v$. By the Comparison Lemma (Khalil, 2002, Lemma 3.4), $V(t)$ is bounded above by the solution of the corresponding equality $\dot{u} = -2K_{\min}u$ which gives $u(t) = V(0)e^{-2K_{\min}t}$. Therefore:

$$V(t) \le V(0)\, e^{-2K_{\min}t}. \tag{3.37}$$

Since $V = \frac{1}{2}\|E\|^2$, substituting and taking the square root yields the tracking error bound directly: Therefore

$$\|E(t)\| \le \|E(0)\|\, e^{-K_{\min}t} \;\longrightarrow\; 0 \quad \text{as } t \to \infty. \tag{3.38}$$

The tracking error converges exponentially to zero at a rate of at least $K_{\min}$. When the error is large the fuzzy gain exceeds $K_{\min}$ and convergence is faster. As the error shrinks the gain reduces smoothly and overshoot is avoided. If bounded disturbances such as roller vibration or wheel slip enter the error dynamics with magnitude $\delta > 0$, the Lyapunov derivative becomes

$$\dot{V} \le -2K_{\min}V + \delta.$$

This inequality satisfies the conditions of Theorem 4.18 in Khalil (2002): $\dot{V}$ is negative whenever $V > \delta/(2K_{\min})$, that is, whenever $\|E\| > \sqrt{\delta/K_{\min}}$. By that theorem, the tracking error is uniformly ultimately bounded and enters the set

$$\|E\| \le \frac{\delta}{K_{\min}}$$

in finite time and remains there. The ultimate bound decreases with $K_{\min}$ and vanishes as $\delta \to 0$, recovering the disturbance-free result. This residual error remains within acceptable bounds for the precision requirements of the Mecabot2.

ROS2 Implementation and Simulation Setup

With the controller design done mathematically, the next step is turning it into working software and setting up the simulation environment. This chapter covers the ROS2 implementation, the Mecabot2 setup in Gazebo and the parameter choices for the simulation run.

ROS2 is the second generation of the Robot Operating System middleware, rebuilt to address the limitations of original ROS in real-time and multi-robot use (Macenski et al., 2022). It uses a publish-subscribe model where software components called nodes exchange typed messages over named topics. This keeps system components decoupled. The trajectory generator, the controller, the odometry estimator and the simulation environment each run as independent nodes. For this work that decoupling means the same controller node can be tested in simulation and later deployed on the real Mecabot2 hardware without code changes, as long as the topic names and message types match.

The controller runs as a single ROS2 node written in Python at a fixed 20 Hz (sampling period $\Delta t = 0.05\,\mathrm{s}$). Each control cycle the node reads the current robot pose $q(t)$ from the `/odom` topic, computes the body-frame tracking error (3.14) from the current desired trajectory point $q_d(t)$, evaluates the three Mamdani fuzzy systems to get adaptive gains K_x, K_y, K_ϕ, applies the control law (3.27)–(3.29) to get the body-frame velocity command ν^* and publishes to the `/cmd_vel` topic. The mecanum inverse kinematics (3.7) is handled by the execution layer below the controller, either the Gazebo mecanum drive plugin in simulation or the embedded base controller on the physical robot. The trajectory clock starts on the first received odometry message, ensuring that the reference trajectory and the robot start position are always synchronised regardless of startup timing.

The ROS2 parameter server lets every tunable quantity be set at launch time through a YAML file or command-line argument without recompiling (Macenski et al., 2022). Exposed parameters include trajectory type, target goal coordinates, trajectory radius and angular speed, minimum and maximum fuzzy output gain values $K_{\min}$ and $K_{\max}$, input normalisation ranges for each error channel and maximum body-frame velocity

magnitudes for output saturation.

The fuzzy inference was written from scratch in Python rather than using an external library. Writing the fuzzification, rule evaluation, aggregation and centroid defuzzification steps directly made it easy to inspect intermediate values during debugging. Keeping it self-contained avoided extra software dependencies that could complicate deployment on the real robot. Although the theoretical formulation in Section 3.3 expresses the crisp output via the continuous centroid integral (3.31), the implementation uses a computationally equivalent singleton form. The three output levels (low, medium and high) are represented directly by the crisp values $k_{\min}$, k_{mid}, and $k_{\max}$ rather than by continuous output membership functions. Defuzzification then reduces to a firing-strength weighted average over those three values:

$$K_i = \frac{\sum_{r=1}^{25} \alpha_r \, z_r}{\sum_{r=1}^{25} \alpha_r}, \tag{4.1}$$

where $\alpha_r = \min(\mu_{A_j}(e_i),\, \mu_{B_k}(\dot{e}_i))$ is the firing strength of rule r and $z_r \in \{k_{\min}, k_{\text{mid}}, k_{\max}\}$ is its output level. When the output membership functions are singletons, this weighted average is algebraically identical to the centroid integral (3.31) (Passino & Yurkovich, 1998). The approach avoids numerical integration entirely, keeping the inference cycle well within the 50 ms control period. If the total firing strength $\sum \alpha_r$ falls below 10^{-12}, the gain defaults to k_{mid} to prevent division by zero.

4.1 Simulation Environment

Gazebo is a three-dimensional rigid-body dynamics simulator tightly integrated with ROS2 through standard plugins (Macenski et al., 2022). The Mecabot2 is described by a URDF file which defines the chassis geometry, wheel positions, inertial parameters and the mecanum drive plugin. The mecanum drive plugin models each wheel as a single contact point with anisotropic friction coefficients chosen to approximate the real mecanum roller array. The Gazebo world is a flat featureless plane with default gravitational acceleration. The robot spawns at the world-frame origin $(x, y, \phi) = (0, 0, 0)$ at the start of each run.

A CSV logging module was added to the controller node to record all relevant signals at 20 Hz throughout each run. The logged quantities include the timestamp, desired and actual positions (x_d, y_d, x, y), the three body-frame tracking errors (E_x, E_y, E_ϕ), the three adaptive gains (K_x, K_y, K_ϕ), the three commanded velocities (V_x, V_y, ω) and the desired velocities (V_{xd}, ω_d). This logged data forms the basis for all result plots in Chapter 5.

4.2 Reference Trajectory and Experimental Design

The circular path experiment uses a radius of $R = 0.20\,\text{m}$. The circular trajectory exercises all three correction channels at once: the robot generates forward velocity, lateral velocity and yaw rate throughout the run. It also gives a steady-state condition with constant desired speed and yaw rate, making it easy to separate transient convergence from steady-state tracking.

The circular path is parameterised as

$$x_d(t) = c_x + R\cos(\omega_{\text{ref}}\, t), \tag{4.2}$$

$$y_d(t) = R\sin(\omega_{\text{ref}}\, t), \tag{4.3}$$

$$\phi_d(t) = \omega_{\text{ref}}\, t + \tfrac{\pi}{2}. \tag{4.4}$$

The desired body-frame velocity comes from differentiating (4.2)–(4.4) and rotating into the body frame:

$$V_{xd}(t) = -R\omega_{\text{ref}}\sin(\omega_{\text{ref}}\, t), \tag{4.5}$$

$$V_{yd}(t) = R\omega_{\text{ref}}\cos(\omega_{\text{ref}}\, t), \tag{4.6}$$

$$\omega_d(t) = \omega_{\text{ref}}. \tag{4.7}$$

Setting $c_x = -R$ ensures the first reference point is at $(c_x + R,\, 0) = (0,\, 0)$, which matches the robot spawn position exactly. This choice eliminates any initial position mismatch between the robot and the trajectory. The robustness of the controller to cases where the robot initial position does not match the desired initial position is demonstrated separately in Section 5.1.4, where the robot spawns with an incorrect heading and successfully converges onto the reference circle. The reference angular speed is set to $\omega_{\text{ref}} = 0.08\,\text{rad/s}$, giving a tangential speed of $R\omega_{\text{ref}} = 0.016\,\text{m/s}$, well within the velocity limits.

The controller gains and fuzzy parameters are listed in Table 4.1. The gain limits were chosen so the minimum gain produces measurable corrective action within one control period and the maximum gain does not saturate the velocity commands for error magnitudes on the 0.20 m radius trajectory.

Table 4.1: Controller and fuzzy system parameters used in the simulation experiments.

Parameter	Symbol	Value
Trajectory radius	R	0.20 m
Trajectory centre	c_x	-0.20 m
Reference angular speed	ω_{ref}	0.08 rad/s
Control frequency	f_c	20 Hz
Min fuzzy gain (x)	$K_{x,\min}$	0.25
Mid fuzzy gain (x)	$K_{x,\text{mid}}$	0.80
Max fuzzy gain (x)	$K_{x,\max}$	2.0
Min fuzzy gain (y)	$K_{y,\min}$	0.18
Mid fuzzy gain (y)	$K_{y,\text{mid}}$	0.60
Max fuzzy gain (y)	$K_{y,\max}$	1.5
Min fuzzy gain (ϕ)	$K_{\phi,\min}$	0.30
Mid fuzzy gain (ϕ)	$K_{\phi,\text{mid}}$	1.00
Max fuzzy gain (ϕ)	$K_{\phi,\max}$	2.5
Max forward velocity	$V_{x,\max}$	0.22 m/s
Max lateral velocity	$V_{y,\max}$	0.22 m/s
Max yaw rate	$\omega_{\max}$	0.80 rad/s

Channel-specific limits are $K_x \in [0.25, 0.80, 2.00]$, $K_y \in [0.18, 0.60, 1.50]$ and $K_\phi \in [0.30, 1.00, 2.50]$ as listed in Table 4.1. Two common failure modes encountered during the development process are also documented in Chapter 5, together with their diagnostic signatures and practical fixes.

The fuzzy membership functions for each input channel are defined over a normalised universe. The forward error E_x and its rate $\dot{E}_x$ are normalised by 0.12 m and 0.12 m/s respectively. The lateral error E_y and its rate $\dot{E}_y$ are normalised by 0.10 m and 0.10 m/s. The heading error E_ϕ and its rate $\dot{E}_\phi$ are normalised by 0.12 rad and 0.12 rad/s. After normalisation, five triangular membership functions are distributed over the universe with centres at $\{-1.0, -0.3, 0.0, 0.3, 1.0\}$ for NB, NS, ZO, PS and PB respectively, with adjacent functions overlapping. The output universe for each channel uses three singleton levels: low ($K_{\min}$), medium (K_{mid}) and high ($K_{\max}$). The channel-specific values are: $K_x \in [0.25, 0.80, 2.00]$, $K_y \in [0.18, 0.60, 1.50]$ and $K_\phi \in [0.30, 1.00, 2.50]$.

4.3 Simulation-to-Real Execution Architecture

The adaptive fuzzy trajectory-tracking controller was implemented as a single ROS2 Python node used for both Gazebo simulation and real-robot operation. The node reads

odometry feedback, generates the desired circular reference online, computes tracking errors in the robot body frame, schedules the adaptive fuzzy gains and publishes velocity commands to `/cmd_vel`. Because the controller is written against standard ROS2 topics rather than a platform-specific API, the same node ran without modification in both environments.

At startup, the node declares all tuning parameters: trajectory type, experiment duration, geometric trajectory parameters, velocity limits, fuzzy gain bounds, normalisation scales and loop period. For the circular experiments, the circle centre is set so that the first reference point coincides with the robot's initial position. An incorrectly placed centre shifts the reference path away from the spawn point and introduces a persistent spatial offset that the controller cannot correct. The loop period is fixed at 0.05 s, giving a 20 Hz control update rate.

Odometry messages update the stored position and yaw. The yaw is recovered from the quaternion orientation, since the controller works in planar coordinates. A key implementation detail is that the internal trajectory clock does not start until the first valid odometry message arrives. This prevents the reference from advancing before a robot state is available and was the direct fix for the phase mismatch observed in earlier experiments, where the reference clock started independently of the actual robot motion.

The desired trajectory is generated online. For the circular case, the reference position and tangent velocity are computed from the circle centre, radius and angular speed at each control step. The tangent velocity direction gives the desired heading via `atan2` and its magnitude gives the desired forward speed. The controller therefore receives a complete reference state (position, heading, forward speed and yaw rate) at every cycle.

Tracking errors are expressed in the robot body frame. The position difference in global coordinates is rotated into body-frame axes to give the forward error E_x and lateral error E_y. The heading error E_ϕ is wrapped into $[-\pi, \pi]$ to avoid the discontinuity that occurs when the desired and actual headings straddle the $\pm\pi$ boundary. Error derivatives are estimated as finite differences over the loop period and used as the second fuzzy input alongside the errors themselves.

Three fuzzy gain blocks run in parallel, one per motion channel, producing K_x, K_y, and K_ϕ. Each block normalises its error and error-rate inputs by channel-specific scale factors before inference. Five triangular membership functions (Negative Big, Negative Small, Zero, Positive Small, and Positive Big) partition the normalised range for each input, giving a 5×5 rule base of 25 Mamdani rules per channel. Rule antecedents are evaluated with the minimum operator. The three output levels (low, medium, and high) map directly to the numerical gain bounds $k_{\min}$, k_{mid}, and $k_{\max}$. Defuzzification uses a weighted average over all fired rules. A fallback to the medium gain applies if the total firing strength is numerically negligible.

The velocity commands combine feedforward and adaptive feedback terms. The for-

ward command adds the desired forward motion projected through the heading error to $K_x E_x$. The lateral command adds the lateral feedforward component to $K_y E_y$. The angular command adds the desired yaw rate to $K_\phi E_\phi$. All three commands are clamped to the platform velocity limits before publication, since large initial errors can temporarily drive unclamped commands outside the safe operating range.

In Gazebo, the `/cmd_vel` output is received by the mecanum drive plugin defined in the robot URDF/Xacro description which applies the commanded velocities to the simulated model and returns odometry to the ROS2 graph. On the physical robot, the same topic is consumed by the bringup layer of the workspace. The bringup chain launches the robot description, sensor filter nodes and the micro-ROS serial bridge that forwards commands to the embedded base controller. Odometry returns over the same path. There is therefore no separate hardware plugin equivalent to the Gazebo plugin: in simulation a virtual execution layer is required; on the physical robot the embedded controller and bringup chain fill that role.

This split is why the simulation-to-real transfer required no changes to the controller. The control law, error formulation, gain-scheduling logic and topic interface stayed fixed. Only the motion-execution backend changed between the two environments.

At every control step the node also writes to a CSV file: time, desired position, actual position, body-frame errors, adaptive gains, commanded velocities and feedforward terms. All trajectory plots, error plots, gain plots and RMSE values reported in Chapter 5 are generated from these logs.

Results and Discussion

This chapter presents the simulation results of the proposed adaptive fuzzy trajectory tracking controller, followed by the hardware experiment results on the physical Mecabot2. The simulation section first documents two failure modes encountered during development, then presents the correctly configured run with full analysis. The hardware section presents the corresponding physical experiment results and discusses the differences from simulation.

5.1 Simulation Results

5.1.1 Failure Mode 1: Wrong Initial Trajectory Parameterisation

The first failure mode arises when the trajectory centre offset c_x does not match the robot spawn position. An initial run was performed with $c_x = 0$, which places the first reference point at $(R,\, 0) = (0.20,\, 0)$ while the robot spawns at the origin. The controller drives the robot toward the reference, but the reference circle is centred at a different location from the one the robot converges to, resulting in a persistent steady-state offset. Figure 5.1 shows the XY trajectory for this case: the actual path settles onto a circle displaced from the desired one, with the same radius but a wrong centre. The spatial displacement visible in the XY trajectory is the geometric cause of the steady-state positioning error. Because the robot converges onto a circle that is permanently offset from the desired one, the distance between the actual and desired positions never decays to zero regardless of how long the controller runs. This is confirmed by Figure 5.2 , which shows the combined planar positioning error $\|E_{xy}\| = \sqrt{(x_d - x)^2 + (y_d - y)^2}$ over time, which measures the Euclidean distance between the desired and actual positions in the xy-plane at each instant. The error rises to approximately 0.36 m and stabilises there.

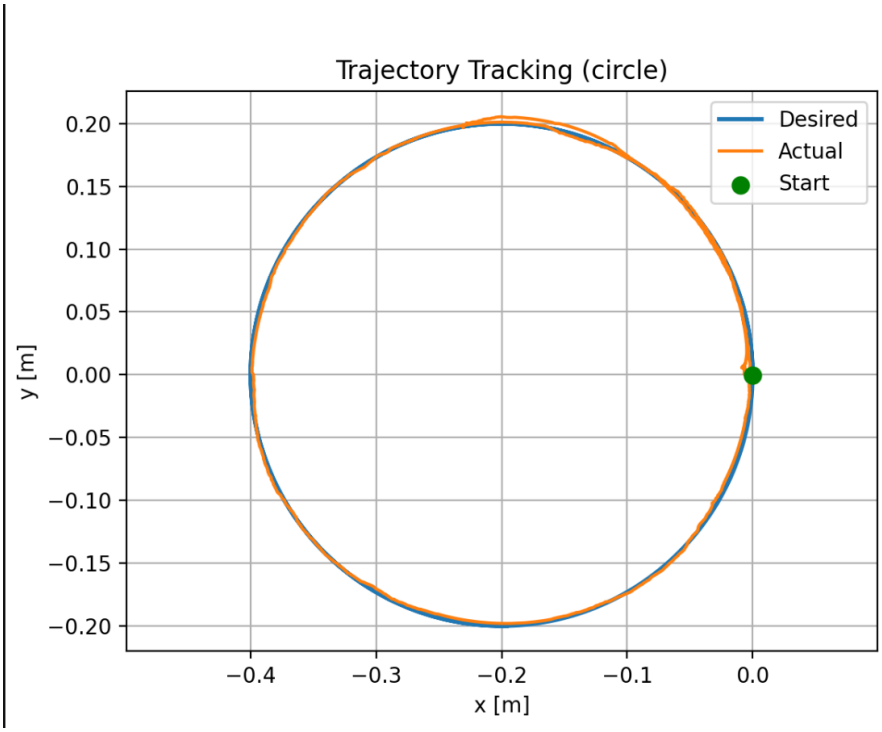


Figure 5.1: Failure Mode 1 - XY trajectory with incorrect centre offset ($c_x = 0$). The actual path (orange) settles onto a circle displaced from the desired path (blue).

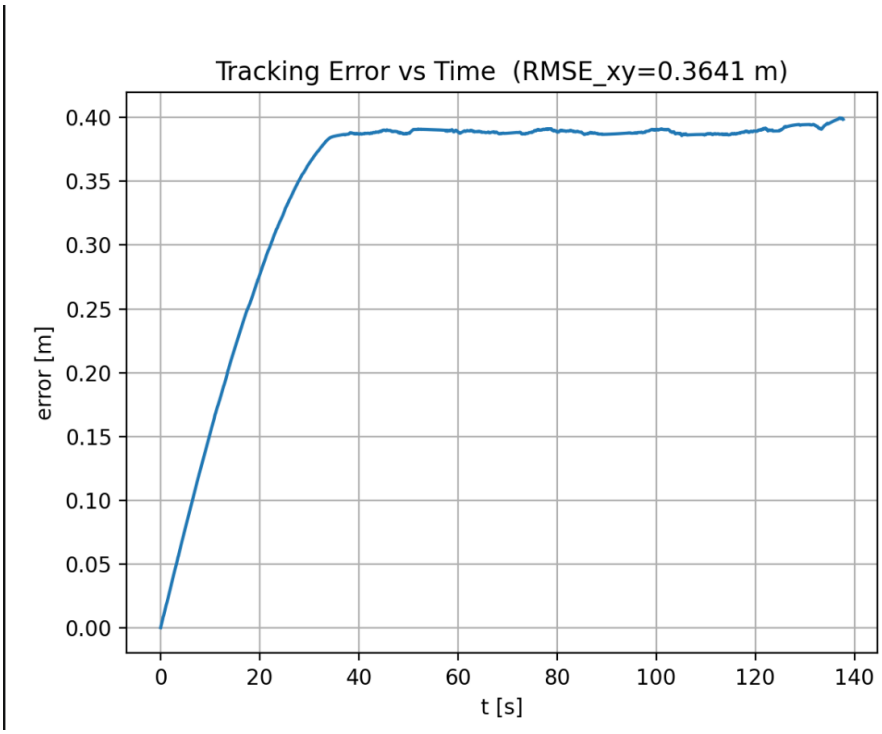
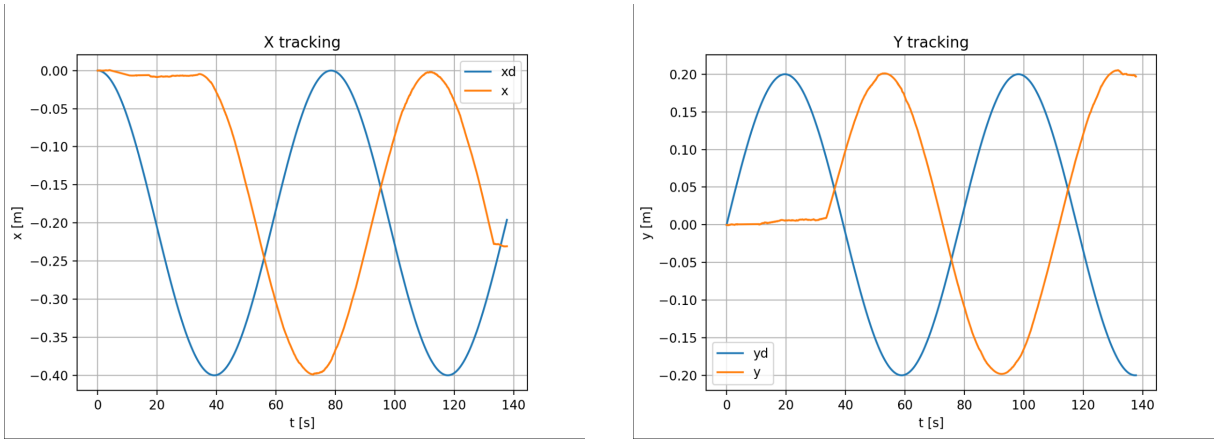


Figure 5.2: Failure Mode 1 - Combined planar positioning error $\sqrt{(x_d - x)^2 + (y_d - y)^2}$ vs time. The error stabilises at a non-zero steady state of approximately 0.36 m. $\text{RMSE}_{xy} = 0.36$ m.



(a) DOF 1 – Forward position (x)

(b) DOF 2 – Lateral position (y)

Figure 5.3: Failure Mode 1 - Position time series showing persistent spatial offset in both channels. Left: forward position x. Right: lateral position y.

Figure 5.3 shows the forward and lateral position time series. Both channels show a clear persistent offset between the desired signal (blue) and the actual signal (orange) throughout the entire run confirming the spatial displacement caused by the incorrect centre offset $c_x = 0$.

The key diagnostic signature is a constant non-zero steady-state error that does not decay further no matter how long the experiment runs. This pattern confirms that the offset comes from a wrong parameter rather than from a disturbance or insufficient gain. The fix is straightforward: set $c_x = -R$ so that $x_d(0) = c_x + R = 0$, matching the robot spawn position exactly. It is important to emphasise that this failure mode is caused entirely by a wrong parameter choice and not by any limitation of the controller itself. The adaptive fuzzy controller is capable of converging onto the desired path even when the robot initial position does not match the desired initial position, as demonstrated in Section 5.1.4. Before describing the second failure mode, it is worth noting that the desired circular path is identical in both failure mode experiments. The difference lies entirely in the actual path: in Failure Mode 1 the actual circle is spatially displaced, while in Failure Mode 2 the actual circle has the correct position but is angularly phase-shifted.

5.1.2 Failure Mode 2: Phase-Lag Due to Odometry Logging Timing Mismatch

The second failure mode arises from a synchronisation error between the odometry logging and the controller startup. In this run the odometry subscriber began recording data before the trajectory controller node finished initialising. As a result, the first several seconds of the log contained odometry samples from when the robot was stationary at the origin, while the reference trajectory had already started advancing. When the controller eventually started, the reference point was already ahead of the robot by several seconds

of trajectory time. The robot then chased a reference that was permanently offset in phase.

Figure 5.4 shows the XY trajectory. Note that the desired circle (blue) is identical to the one in Figure 5.1 - this is intentional, since the same trajectory parameters are used in both failure mode experiments. The difference lies entirely in the actual path (orange): unlike Failure Mode 1 where the actual circle was spatially displaced, here the actual circle is centred correctly and has the correct radius, confirming that the trajectory parameterisation itself is sound. However, the actual path is angularly shifted from the desired one: the robot is always at approximately the correct radius but at the wrong angular position on the circle.

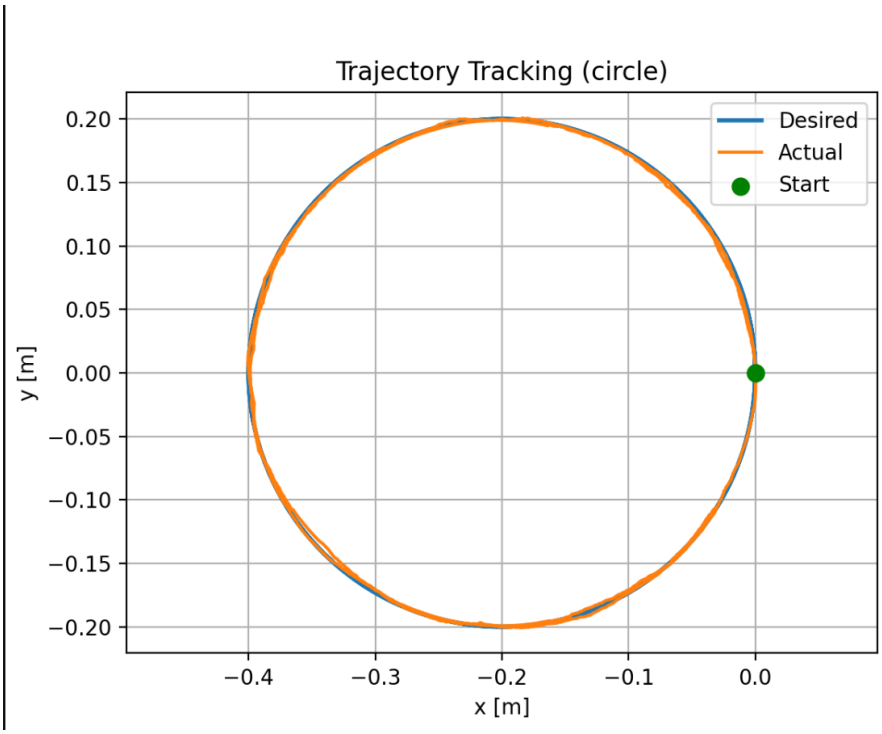
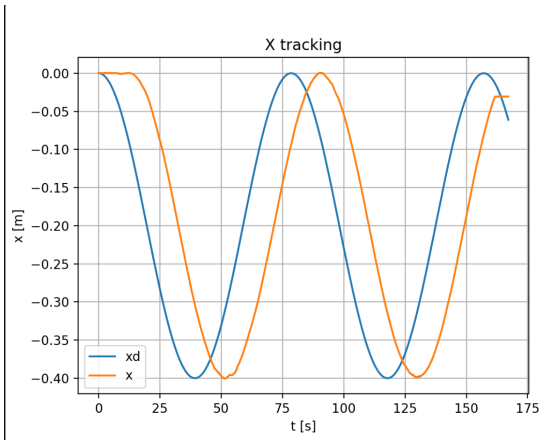
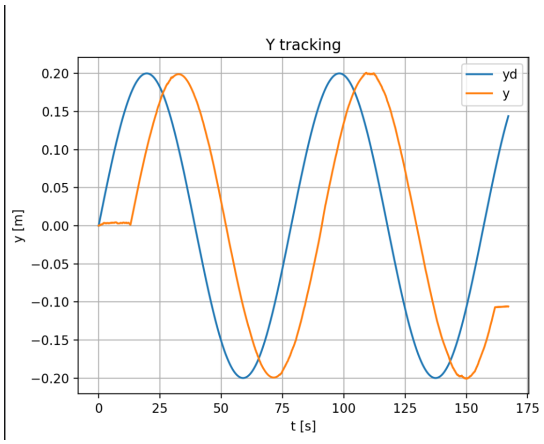


Figure 5.4: Failure Mode 2 - XY trajectory with odometry logging started before the controller. The actual path (orange) has the correct radius and centre but is angularly phase-shifted from the desired circle (blue). The green dot marks the robot spawn position.

Figure 5.5 shows the x and y position time series respectively. Both channels show a clear horizontal phase shift between the desired signal (blue) and the actual signal (orange) that persists for the entire duration of the run. The amplitude and shape of the two signals match closely, which rules out a gain or bandwidth problem. The shift is approximately constant over time, consistent with a fixed timing offset introduced at startup rather than a growing instability.



(a) DOF 1 - Forward position (x)



(b) DOF 2 - Lateral position (y)

Figure 5.5: Failure Mode 2 - Position time series showing persistent phase shift in both channels caused by the logging timing mismatch. Left: forward position x. Right: lateral position y. Both channels show a clear horizontal phase shift between the desired signal (blue) and the actual signal (orange).

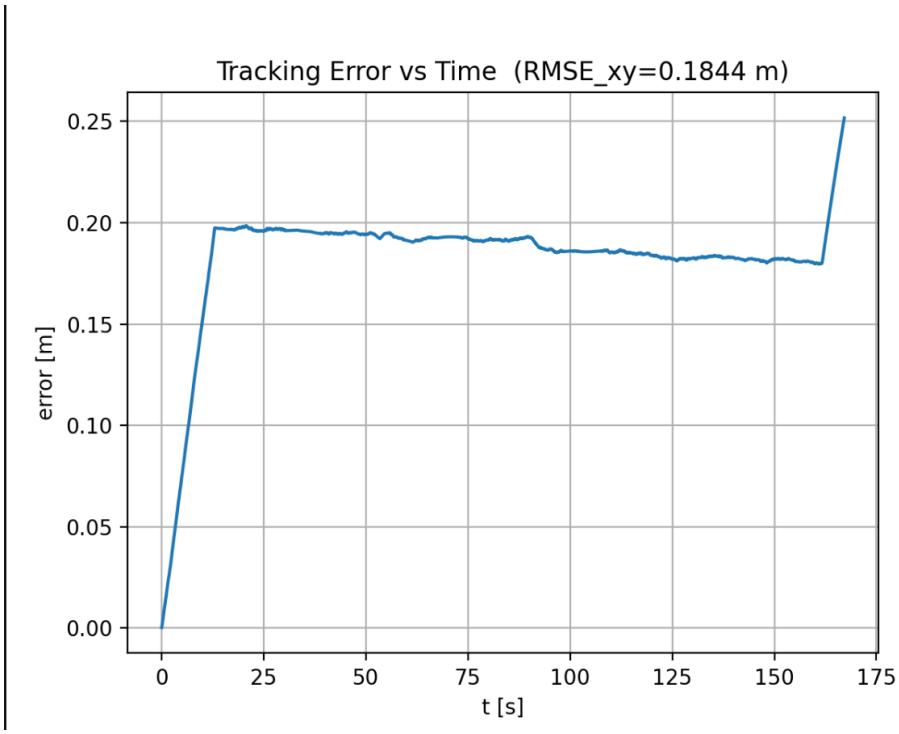


Figure 5.6: Failure Mode 2 - Combined planar positioning error $\sqrt{(x_d - x)^2 + (y_d - y)^2}$ vs time. Persistent oscillation at the trajectory frequency confirms a fixed timing offset rather than a transient. $\text{RMSE}_{xy} = 0.1844\,\text{m}$.

Figure 5.6 shows the combined planar positioning error $\sqrt{(x_d - x)^2 + (y_d - y)^2}$ over time. The error oscillates at the trajectory frequency and does not decay, which confirms that the controller is not converging to the reference. A converging error would decrease monotonically; this one does not. The RMSE is 0.1844 m.

The diagnostic signature is therefore distinct from Failure Mode 1. In Failure Mode 1 the error settles to a constant non-zero value that reflects a spatial offset. In Failure Mode 2 the error oscillates periodically, reflecting a temporal offset. The XY trajectory also provides a clear visual distinction: in Failure Mode 1 the actual circle is displaced spatially from the desired one, while in Failure Mode 2 the actual circle coincides with the desired one in shape and centre but is rotated by a fixed angle.

The fix was implemented in the logging script `plot_odom_txt.py` by aligning the start of the odometry record with the first control command issued by the controller node. The trajectory clock was also moved so that it starts on the first odometry message received after the controller has finished initialising, ensuring that the reference trajectory and the robot position are always synchronised regardless of node startup order.

The control signals, adaptive gains and body-frame error time series are not presented for this failure mode run. These runs were conducted as early diagnostic experiments during the development phase, before the full CSV logging pipeline was integrated into the controller node. Since the primary purpose of these runs is to document and diagnose the failure behaviour rather than to analyse the controller performance in detail, the trajectory and positioning error plots presented above are sufficient to characterise each failure mode completely. The detailed signal analysis is reserved for the correctly configured run in Section 5.1.3, where all signals are logged and analysed in full.

5.1.3 Correctly Configured Simulation Run

With both corrections applied, the simulation run uses $c_x = -0.20\,\text{m}$ and $\omega_{\text{ref}} = 0.08\,\text{rad/s}$. All other parameters are as listed in Table 4.1. The controller was run for 150 s and all signals were logged at 20 Hz, giving 2941 samples.

Trajectory tracking. Figure 5.7 shows the XY trajectory. The robot starts at the green dot at (0, 0), which is the first point on the desired circle. A short transient is visible at the beginning as the fuzzy gains increase to correct the initial heading error, after which the actual path lies essentially on top of the desired circle for the remainder of the run.

Position error. Figure 5.8 shows the positioning error over time. The error peaks at approximately 0.014 m during the initial transient as the controller corrects the starting heading error, then decays to a steady-state level of approximately 0.002 m within the first ten seconds. This initial peak and subsequent monotonic decay match the exponential convergence bound from Equation (3.38). The steady-state residual is consistent with the theoretical bound $\delta/K_{\min}$ from Section 3.3, where δ represents the unmodelled roller contact disturbances in the Gazebo simulation. The overall RMSE over the full 150 s run is 0.0026 m.

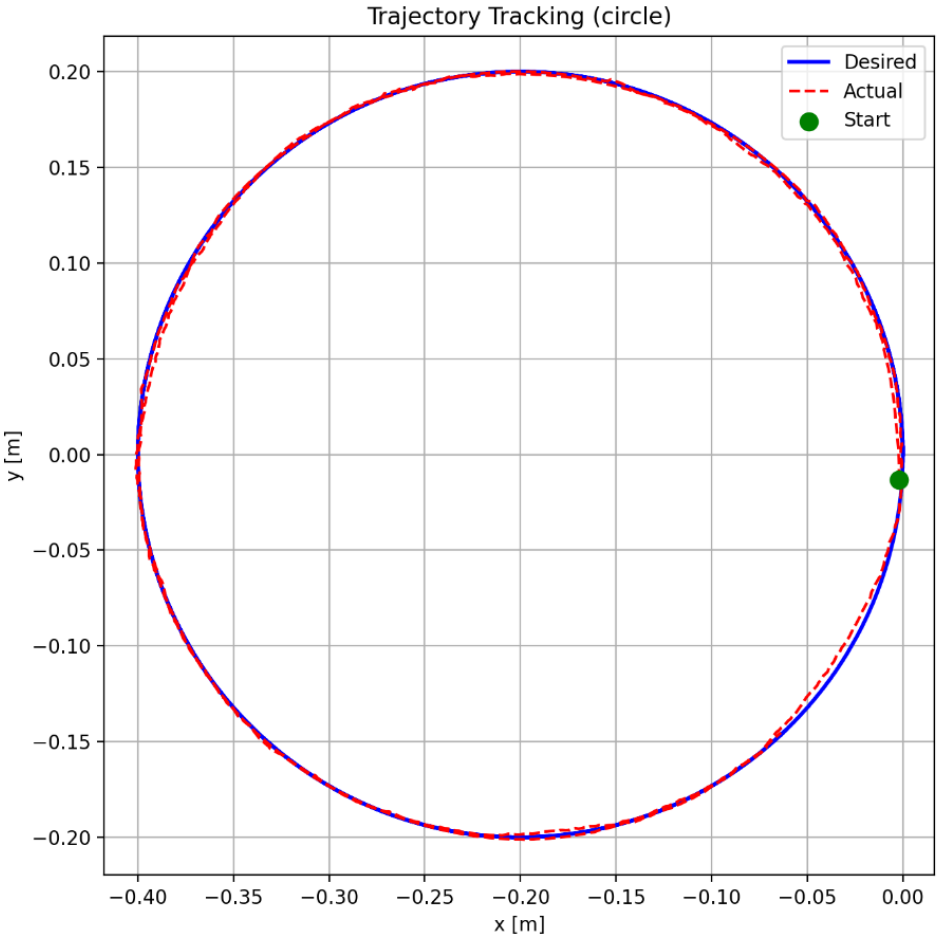


Figure 5.7: Simulation - XY trajectory with correct configuration ($c_x = -0.20\,\text{m}$, $\omega_{\text{ref}} = 0.08\,\text{rad/s}$). The actual path (red dashed) converges onto the desired circle (blue) within the first quarter revolution. The green dot marks the robot start position.

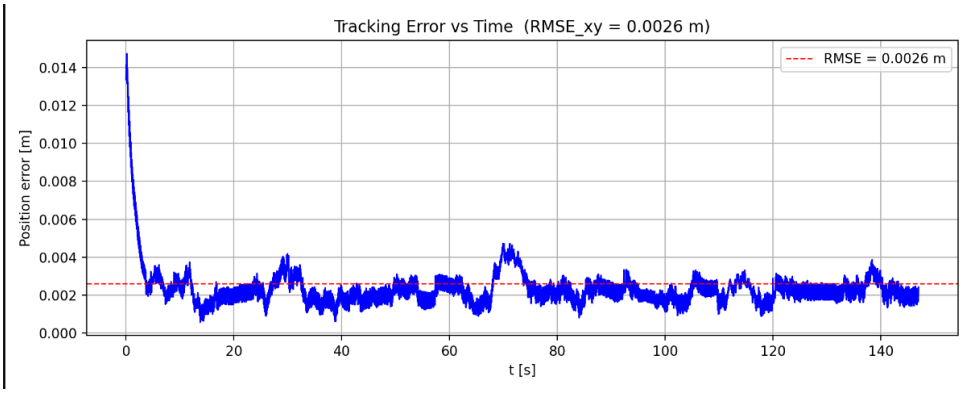


Figure 5.8: Simulation - Positioning error vs time. The initial transient peak of 0.014 m decays to a steady-state level of approximately 0.002 m. The red dashed line marks the overall $\text{RMSE}_{xy} = 0.0026\,\text{m}$.

Position time series.

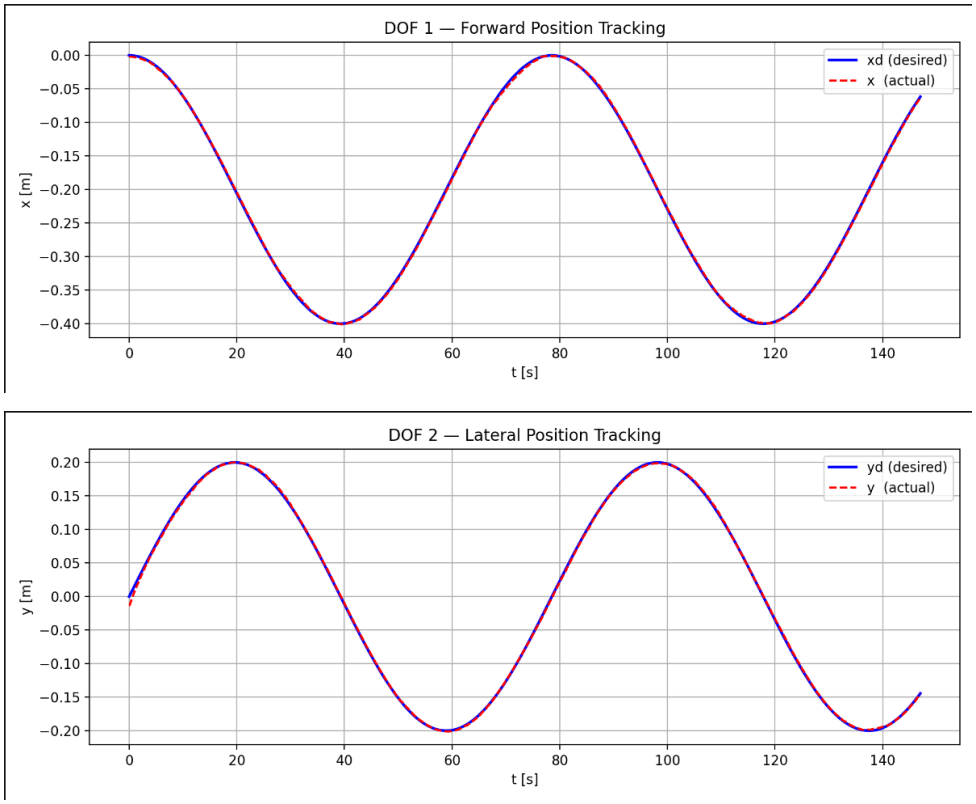


Figure 5.9: Simulation position tracking. Top: DOF 1 forward position tracking, desired x_d (blue) and actual x (red dashed). Bottom: DOF 2 lateral position tracking, desired y_d (blue) and actual y (red dashed). Both channels show close agreement throughout the run.

Figure 5.9 shows the forward and lateral position time series, with the desired and actual signals overlaid. Both channels show excellent agreement throughout the run with negligible phase offset confirming that the trajectory clock synchronisation is correct and that neither DOF lags the reference after the initial transient has settled.

Control signals and three-DOF operation. Figure 5.10 shows the three commanded velocity signals over time. The forward velocity V_x closely follows the sinusoidal desired profile V_{xd}, confirming that DOF 1 is tracked accurately. The lateral velocity V_y and the yaw rate ω are simultaneously active throughout the run, demonstrating that the controller is managing all three degrees of freedom concurrently. This simultaneous three-channel operation is the fundamental advantage of the mecanum platform: unlike a differential-drive robot, the lateral velocity channel can be used independently without interfering with forward motion or rotation. The small residual noise visible in all three channels reflects the roller contact disturbances modelled by Gazebo.

Adaptive gains. Figure 5.11 shows the three adaptive gains K_x, K_y and K_ϕ over time. All three gains are elevated during the initial transient period while the controller corrects the starting heading error, then settle to lower steady-state values once the robot

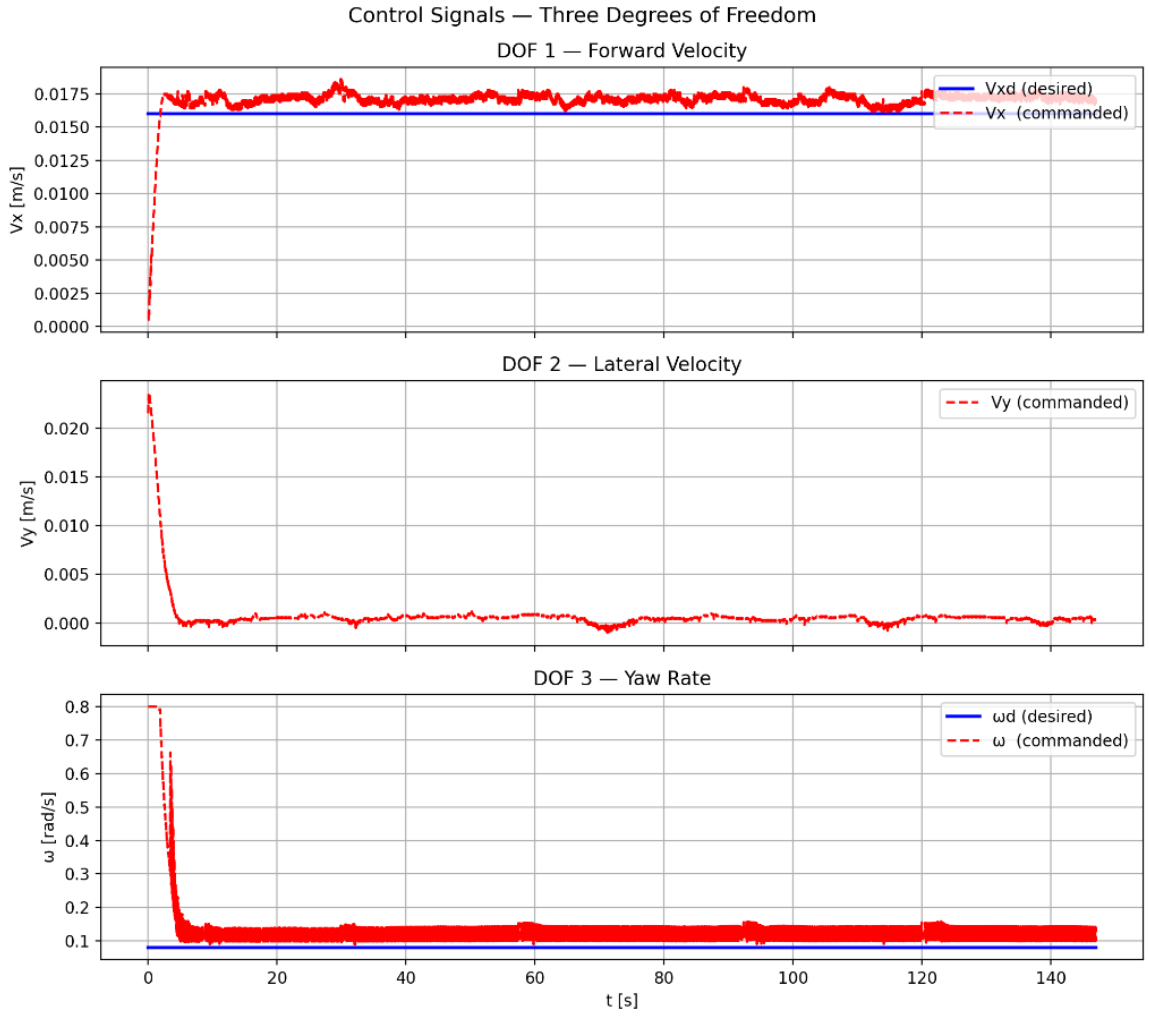


Figure 5.10: Simulation - Control signals for all three degrees of freedom. Top: DOF 1 forward velocity V_x vs desired V_{xd}. Middle: DOF 2 lateral velocity V_y (commanded). Bottom: DOF 3 yaw rate ω vs desired ω_d. All three channels are active simultaneously, demonstrating full holonomic operation.

is tracking the circle accurately. This behaviour is exactly what the fuzzy rule base encodes: large error produces large gain and small converging error produces small gain. The heading gain K_ϕ shows the largest initial spike, which is expected since the heading error E_ϕ is the dominant component of the initial mismatch. The fact that all gains decrease and stabilise rather than growing unboundedly is consistent with the Lyapunov stability proof, which guarantees bounded gains throughout operation.

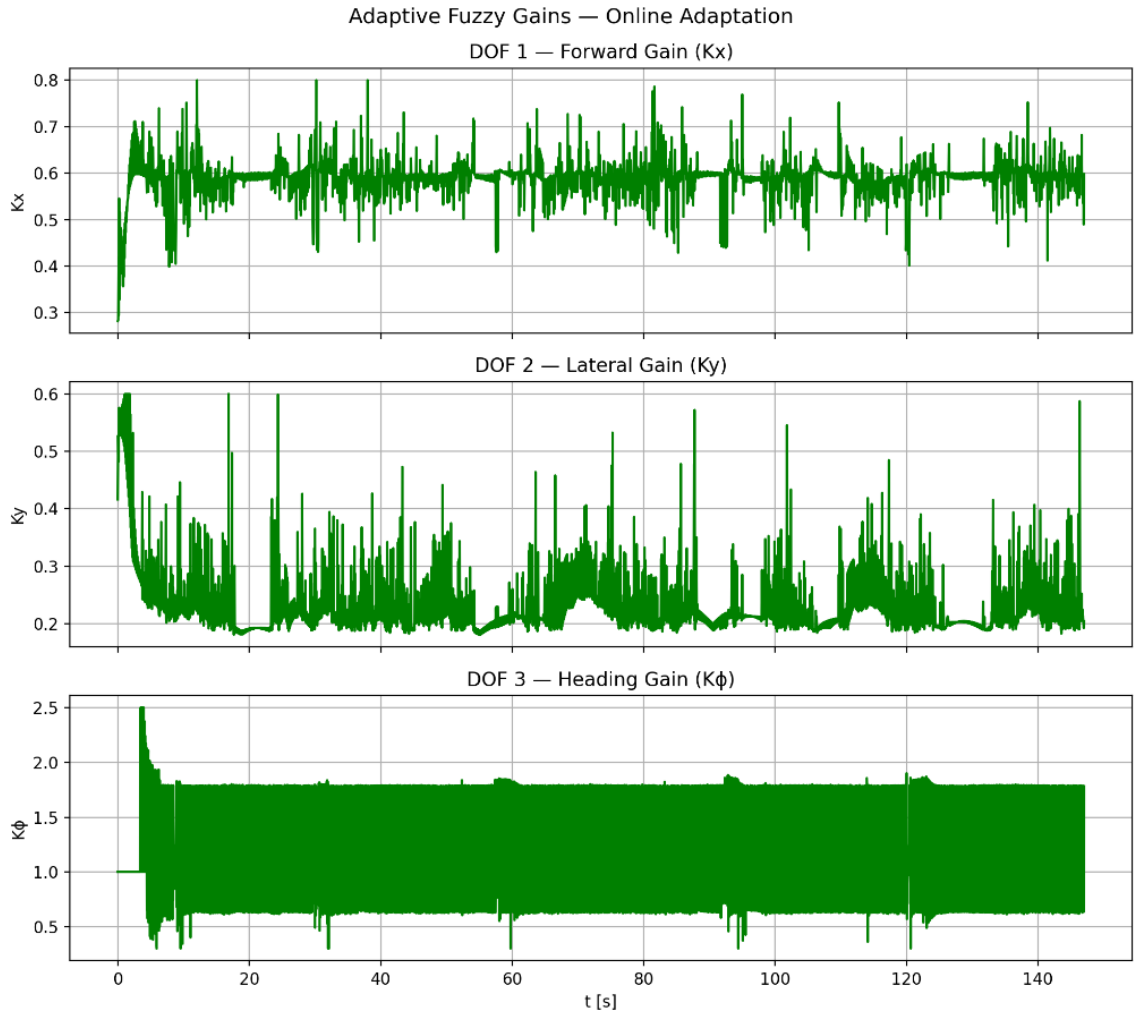


Figure 5.11: Simulation - Adaptive fuzzy gains for all three channels. Top: K_x (forward). Middle: K_y (lateral). Bottom: K_ϕ (heading). Gains are elevated during the initial transient and settle to lower steady-state values as the robot converges onto the reference circle.

Body-frame errors. Figure 5.12 shows the three body-frame tracking errors E_x, E_y and E_ϕ over time. The heading error E_ϕ exhibits the clearest exponential decay, dropping from approximately 1.5 rad at $t = 0$ to near zero within the first ten seconds. This decay profile visually demonstrates the exponential convergence bound established by Equation (3.38). The forward and lateral errors E_x and E_y remain small throughout the run, fluctuating near zero at the level of the Gazebo roller noise.

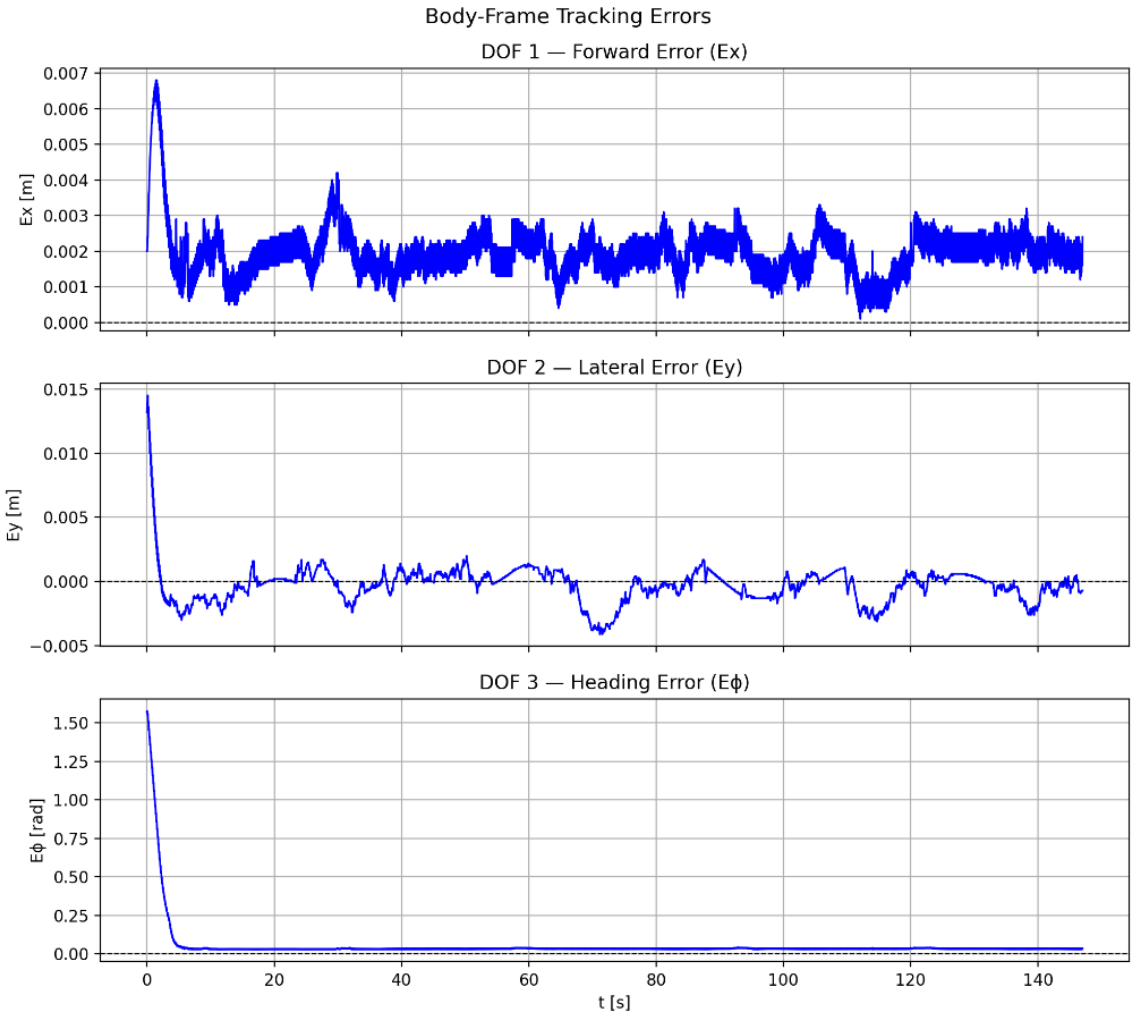


Figure 5.12: Simulation - Body-frame tracking errors. Top: E_x (forward). Middle: E_y (lateral). Bottom: E_ϕ (heading). The heading error decays exponentially from the initial value, directly illustrating the convergence bound of Equation (3.38).

5.1.4 Simulation Run: Robot Spawning Outside the Circle

An additional simulation experiment was conducted to test the controller recovery from a large initial displacement. The robot spawns at the world-frame origin (0, 0) while the trajectory centre remains at $c_x = -0.20\,\text{m}$. The spawn point lies on the circumference of the desired circle but the initial heading $\phi(0) = 0$ does not match the desired heading $\phi_d(0) = \pi/2$. The controller must therefore correct a simultaneous heading and lateral error before converging onto the reference path. All other parameters are as listed in Table 4.1.

Trajectory tracking. Figure 5.13 shows the XY trajectory. The robot starts at the green dot at (0, 0). A visible convergence arc is traced in the first few seconds before the actual path lies essentially on top of the desired circle for the remainder of the run.

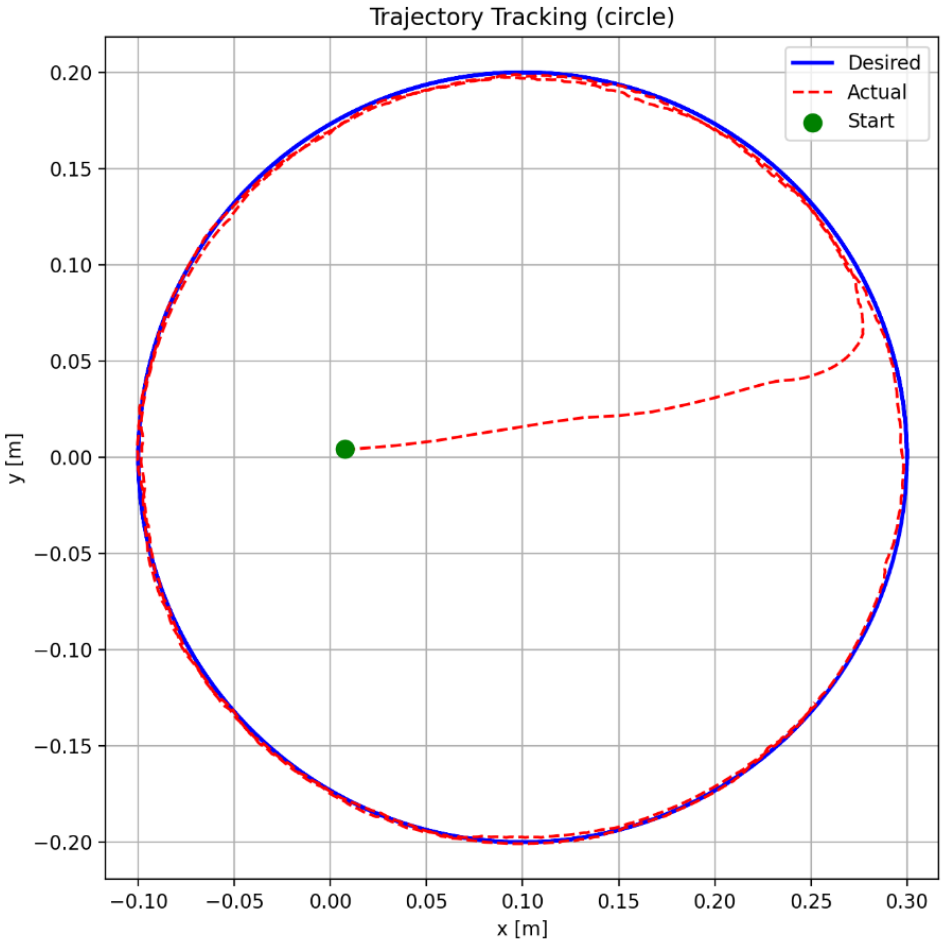


Figure 5.13: Simulation (outside spawn) - XY trajectory. The robot spawns at (0, 0) on the circumference of the desired circle (blue) with incorrect heading. The actual path (red dashed) traces a short convergence arc before locking onto the reference circle. The green dot marks the robot start position.

Position error. Figure 5.14 shows the positioning error over time. The error peaks at approximately 0.30 m during the initial transient as the controller corrects the com-

bined heading and lateral displacement, then decays monotonically to a steady-state level comparable to the correctly initialised run. The overall RMSE over the full 150 s run is 0.0198 m. The elevated RMSE relative to the correctly initialised run (0.0026 m) is entirely due to the large initial transient contribution. The steady-state residual is comparable once convergence is complete.

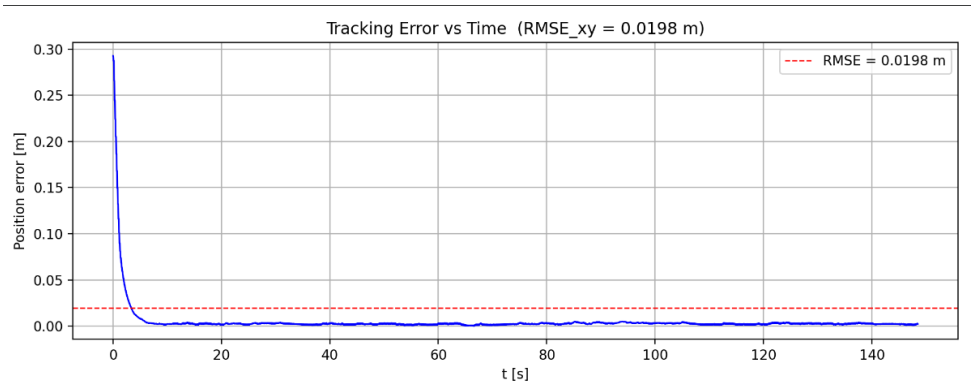


Figure 5.14: Simulation (outside spawn) - Combined planar positioning error $\sqrt{(x_d - x)^2 + (y_d - y)^2}$ vs time. The initial peak of approximately 0.30 m decays monotonically as the controller converges onto the circle. The red dashed line marks the overall $\text{RMSE}_{xy} = 0.0198$ m.

Position time series. Figure 5.15 shows the forward and lateral position time series, with the desired and actual signals overlaid. Both channels show a clear initial mismatch followed by convergence to the sinusoidal reference profile. After the transient both channels agree closely with negligible phase offset.

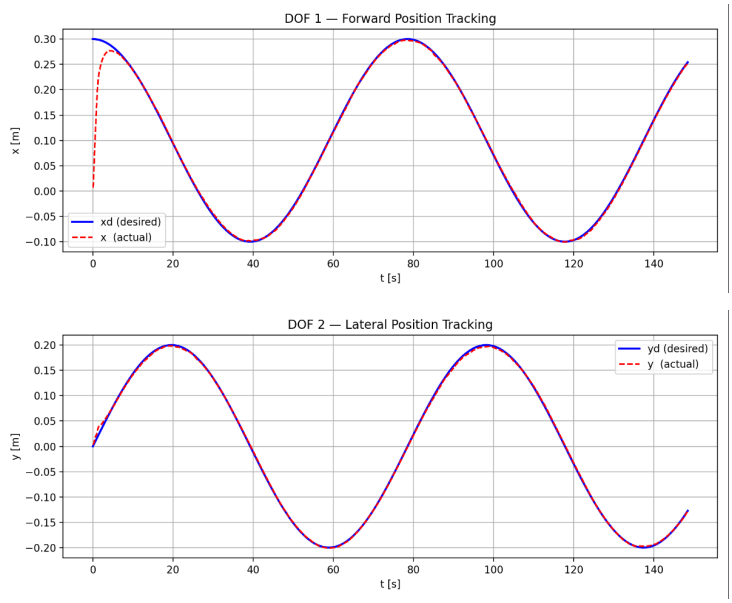


Figure 5.15: Simulation (outside spawn) - Position time series. Top: DOF 1 forward position tracking, desired x_d (blue) and actual x (red dashed). Bottom: DOF 2 lateral position tracking, desired y_d (blue) and actual y (red dashed). Both channels converge from the initial displacement to close agreement within approximately 20 s.

Control signals and three-DOF operation. Figure 5.16 shows the three commanded velocity signals over time. All three channels exhibit large initial spikes as the fuzzy gains respond to the large starting errors. The forward velocity V_x and yaw rate ω both reach near-saturation briefly before settling to their steady-state profiles. After the transient all three channels operate normally with no persistent oscillation, demonstrating stable recovery under a large initial displacement.

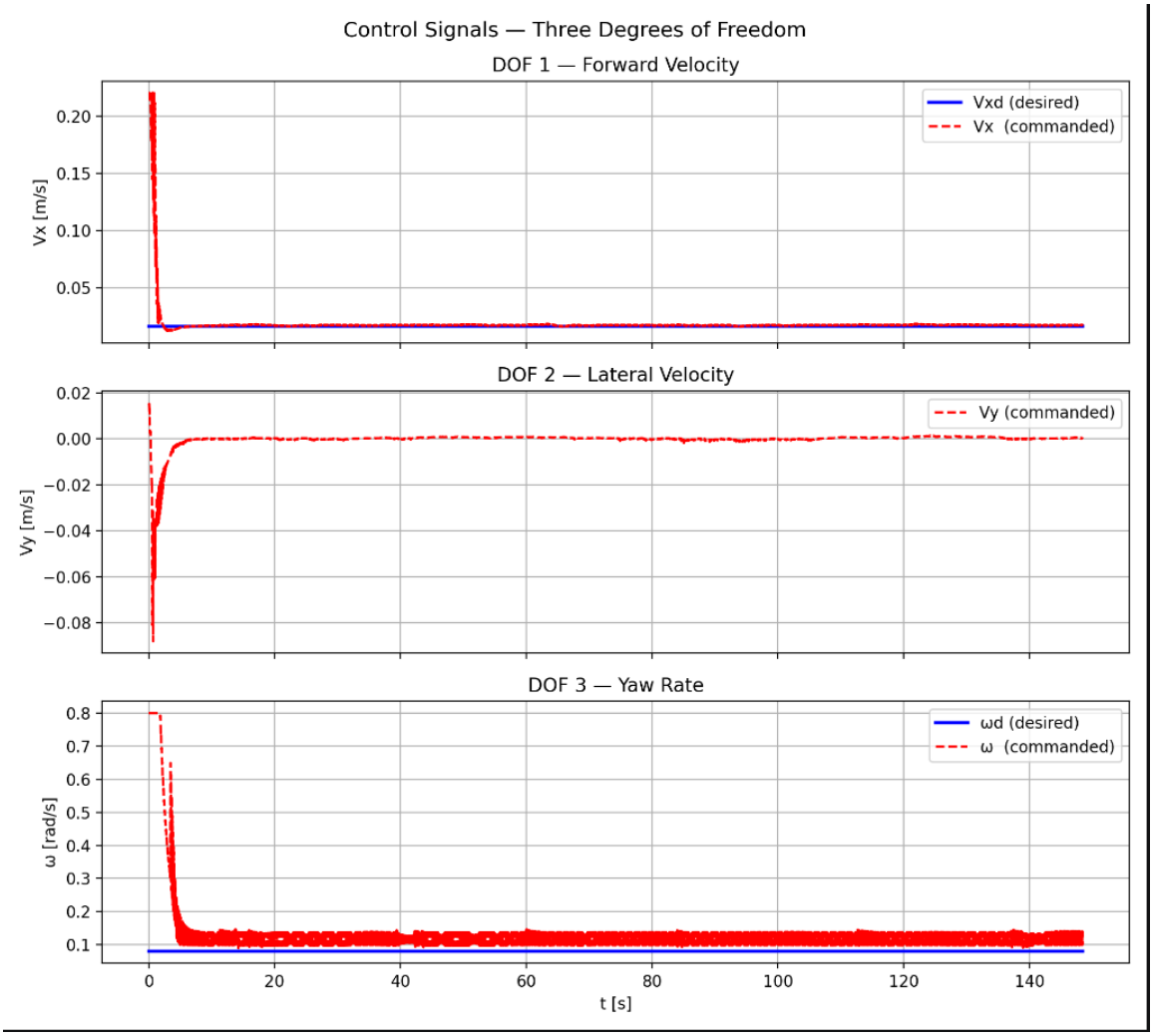


Figure 5.16: Simulation (outside spawn) - Control signals for all three degrees of freedom. Top: DOF 1 forward velocity V_x vs desired V_{xd}. Middle: DOF 2 lateral velocity V_y (commanded). Bottom: DOF 3 yaw rate ω vs desired ω_d. All three channels show large initial transients followed by normal steady-state operation.

Adaptive gains. Figure 5.17 shows the three adaptive gains K_x, K_y and K_ϕ over time. All three gains are elevated toward $K_{\max}$ at the start of the run in response to the large initial errors, then decay as the robot converges onto the circle. This behaviour directly illustrates the adaptive mechanism encoded in the fuzzy rule base: large error produces large gain and small converging error produces small gain. All gains remain within $[K_{\min}, K_{\max}]$ throughout the run, consistent with Equation (3.35).

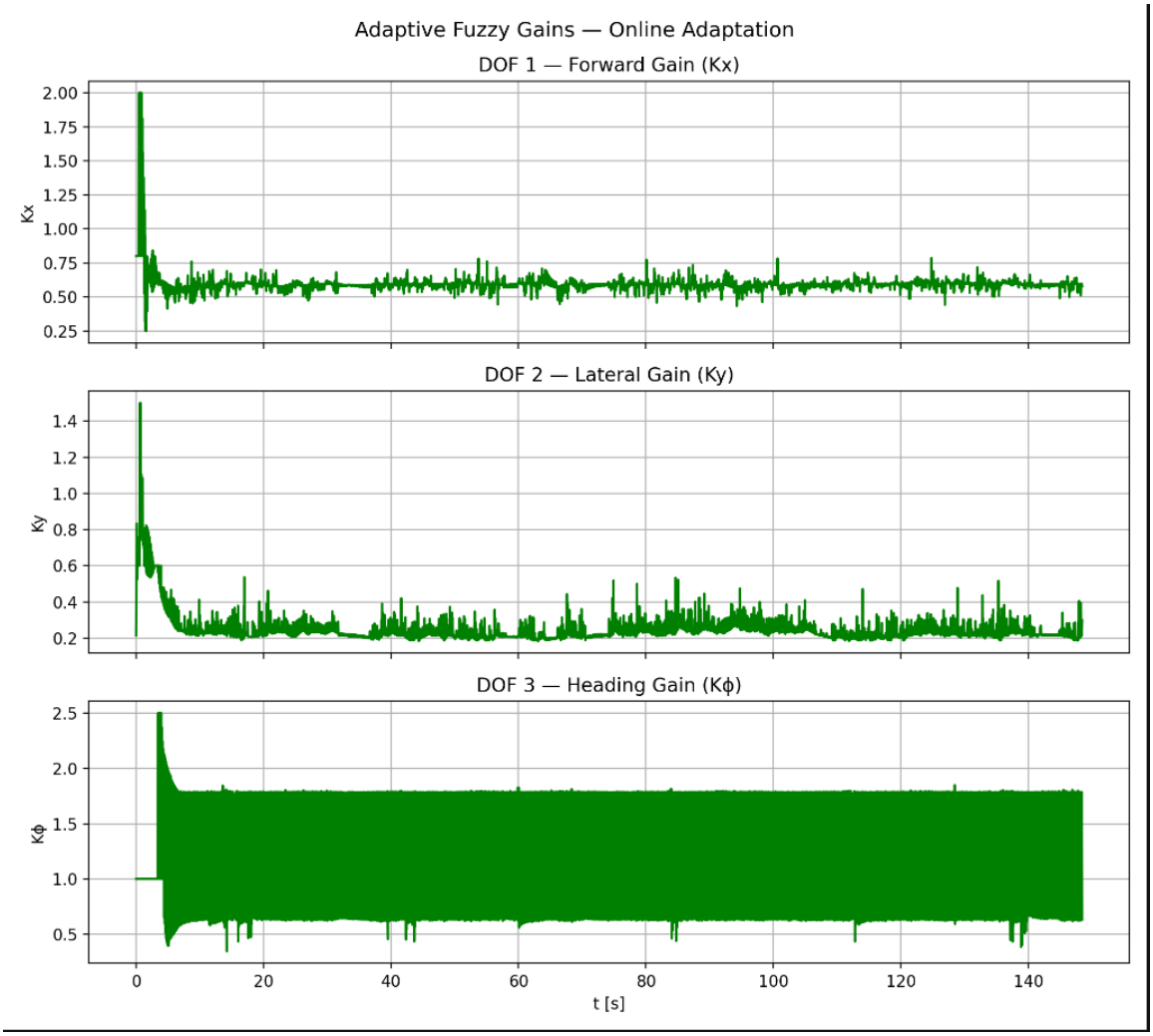


Figure 5.17: Simulation (outside spawn) - Adaptive fuzzy gains for all three channels. Top: K_x (forward). Middle: K_y (lateral). Bottom: K_ϕ (heading). All gains spike toward $K_{\max}$ during the initial convergence phase and settle to lower steady-state values once the robot is tracking the reference circle.

Body-frame errors. Figure 5.18 shows the three body-frame tracking errors E_x, E_y and E_ϕ over time. The heading error E_ϕ dominates the initial transient, decaying exponentially from approximately $\pi/2$ rad to near zero within the first ten seconds, directly reproducing the convergence behaviour predicted by Equation (3.38). The forward and lateral errors also decay from their initial values and settle at the same near-zero steady-state level as in the correctly initialised run.

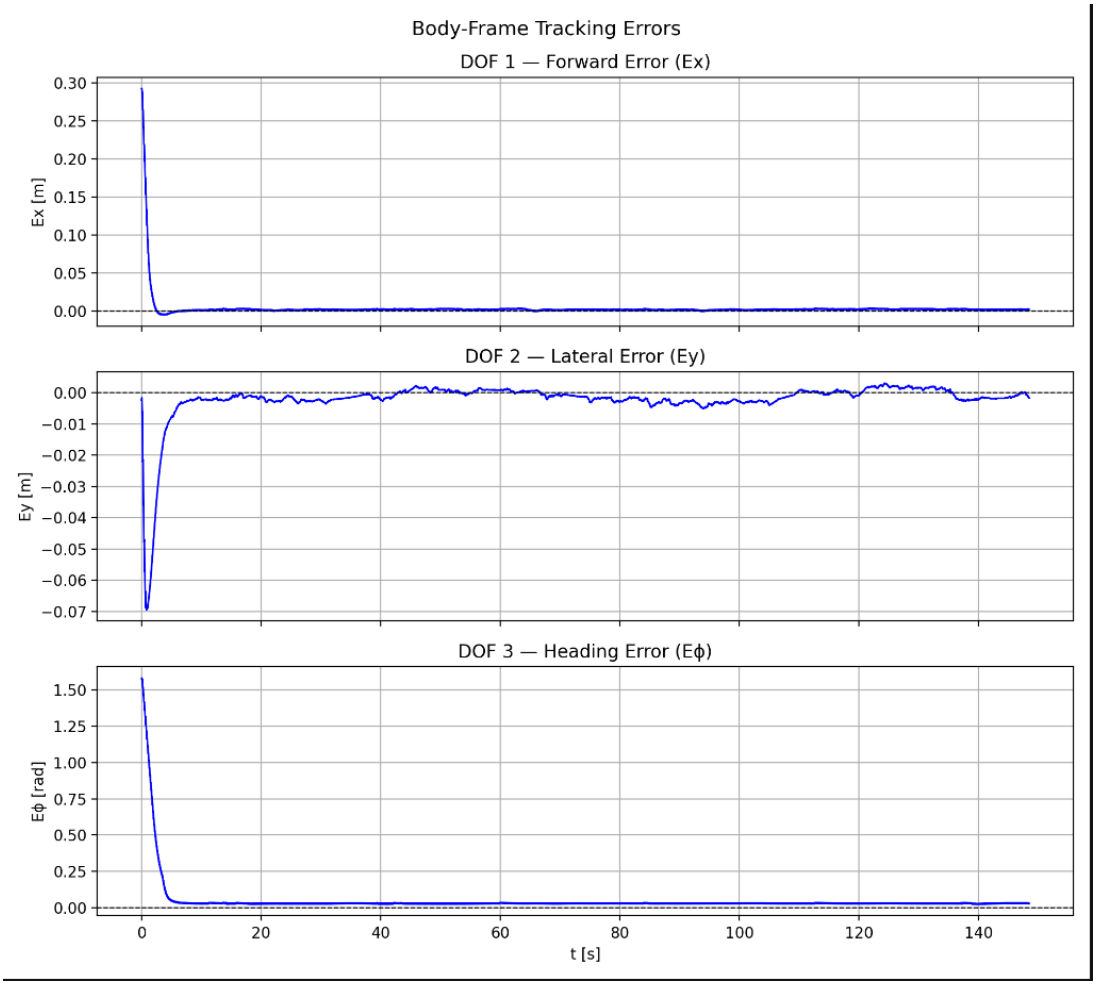


Figure 5.18: Simulation (outside spawn) - Body-frame tracking errors. Top: E_x (forward). Middle: E_y (lateral). Bottom: E_ϕ (heading). All three errors decay from their initial values to near-zero steady state, consistent with the exponential convergence bound of Equation (3.38).

This experiment confirms that the adaptive fuzzy controller converges from a large initial displacement without oscillation or instability. The higher overall RMSE of 0.0198 m compared to the correctly initialised run is entirely attributable to the initial transient and not to a deficiency in steady-state tracking performance.

5.1.5 Comparison of Simulation Runs

The simulation runs together illustrate how initial conditions affect convergence behaviour independently of the controller design. In the correctly initialised run (Section 5.1.3) the robot spawns at (0, 0), which coincides exactly with the first point on the desired circle. Only a heading error is present at startup: $E_x(0) \approx 0$, $E_y(0) \approx 0$ and $E_\phi(0) = \pi/2$. The fuzzy gain scheduler responds by elevating only K_ϕ while K_x and K_y remain near their minimum values. The robot therefore stays close to the circle throughout and the convergence arc is small, producing a smooth trajectory with $\text{RMSE}_{xy} = 0.0026\,\text{m}$.

In the outside spawn run (Section 5.1.4) the robot spawns at the same point (0, 0) but with an incorrect initial heading $\phi(0) = 0$ instead of the desired $\phi_d(0) = \pi/2$, while the spawn point remains on the circumference of the circle. All three body-frame errors are non-zero at startup: $E_x(0) \neq 0$, $E_y(0) \neq 0$ and $E_\phi(0) \neq 0$. The fuzzy rule base drives all three gains toward $K_{\max}$ simultaneously, producing a large initial transient and a visible convergence arc in the XY trajectory. Once the robot reaches the circle the steady-state tracking quality is indistinguishable from the correctly initialised run, but the large initial transient raises the overall RMSE to 0.0198 m.

The key point is that the difference in trajectory smoothness between the two runs is not caused by the controller but by the initial Lyapunov energy $V(0) = \frac{1}{2}(E_x^2 + E_y^2 + E_\phi^2)$. In the correctly initialised run only $E_\phi(0)$ contributes to $V(0)$, so the initial energy is small and convergence is fast and smooth. In the outside spawn run all three terms contribute, the initial energy is much larger and the controller must dissipate it before steady-state tracking can begin. This is consistent with the exponential convergence bound of Equation (3.38), which predicts that a larger $\|E(0)\|$ produces a longer transient at the same convergence rate $K_{\min}$.

5.2 Hardware Experiment Results

Hardware experiments were conducted on the physical Mecabot2 using the same ROS2 controller node and parameter file as in the simulation. The same circular trajectory with $R = 0.20\,\text{m}$, $c_x = -0.20\,\text{m}$ and $\omega_{\text{ref}} = 0.08\,\text{rad/s}$ was commanded. Odometry was recorded from the onboard wheel encoders via the `/odom` topic. Because the controller node was unchanged, the hardware results reflect the same high-level control law operating through a different execution backend. Performance differences relative to simulation (higher RMSE, more gain fluctuation, noisier error signals) arise from the physical execution layer: wheel slip, roller vibration, odometry quantisation and embedded actuation delays that the Gazebo model does not replicate.

Trajectory tracking. Figure 5.19 shows the XY trajectory recorded on the physical Mecabot2. The robot converges onto the desired circle within the first quarter revolution, consistent with the simulation behaviour. The actual path follows the reference closely throughout the run. Minor deviations along the circle perimeter are consistent with roller vibration and floor friction effects expected on a physical platform.

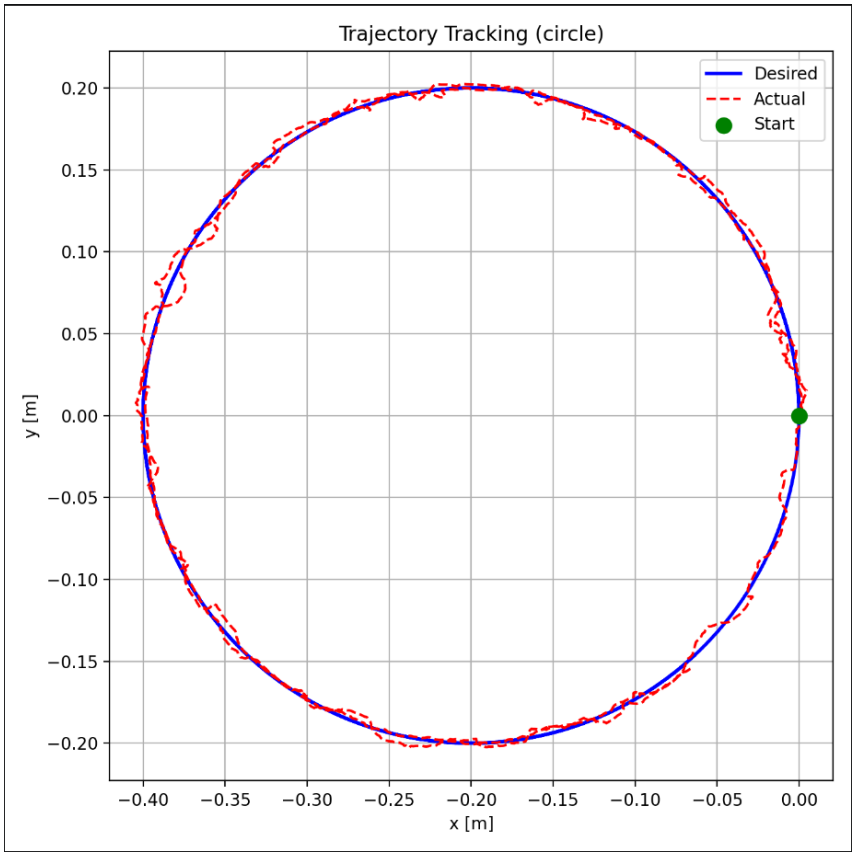


Figure 5.19: Hardware - XY trajectory. The actual path (red dashed) converges onto the desired circle (blue). The green dot marks the robot start position.

Position error. Figure 5.20 shows the positioning error over time. The error settles

quickly into a noisy steady-state fluctuation centred near zero, reflecting roller-ground contact noise from the physical floor surface. The overall RMSE over the full run is 0.0046 m, approximately 1.8× higher than the simulation RMSE of 0.0026 m. This increase is expected: real roller vibration, wheel slip and odometry drift are absent from the Gazebo contact model. The error remains bounded and small throughout the entire run, confirming stable tracking on the physical platform.

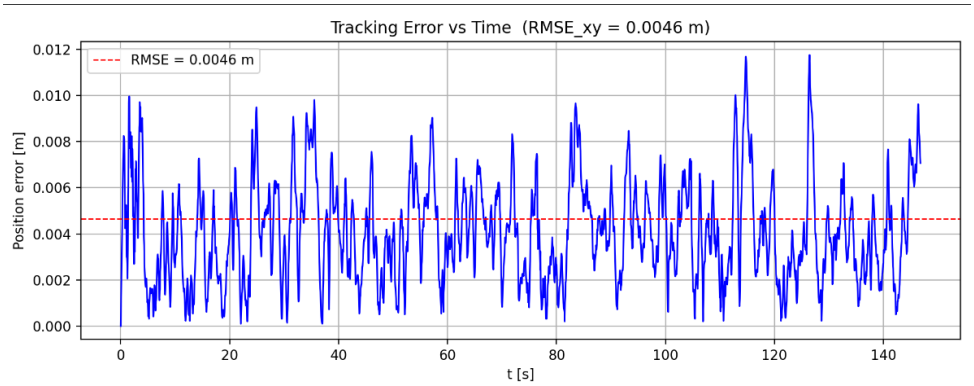


Figure 5.20: Hardware - Positioning error vs time. The error fluctuates in a noisy steady state throughout the run. The red dashed line marks the overall $\text{RMSE}_{xy} = 0.0046\,\text{m}$.

Position time series. Figure 5.21 shows the forward and lateral position time series with the desired and actual signals overlaid. Both channels track the sinusoidal reference closely with negligible phase offset, confirming that the trajectory clock synchronisation works correctly on real hardware.

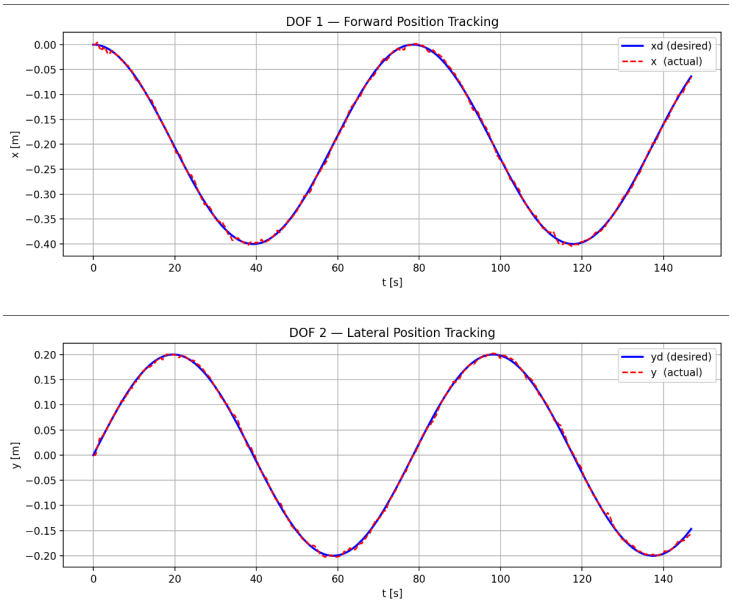


Figure 5.21: Hardware - Position time series. Top: DOF 1 forward position tracking, desired x_d (blue) and actual x (red dashed). Bottom: DOF 2 lateral position tracking, desired y_d (blue) and actual y (red dashed). Both channels show close agreement throughout the run.

Control signals and three-DOF operation. Figure 5.22 shows the three commanded velocity signals on the physical robot. The forward velocity V_x tracks the desired profile V_{xd} closely. The lateral velocity V_y and yaw rate ω are simultaneously active throughout the run, confirming full three-channel mecanum operation on real hardware. Higher noise compared to simulation reflects real encoder and floor disturbances.

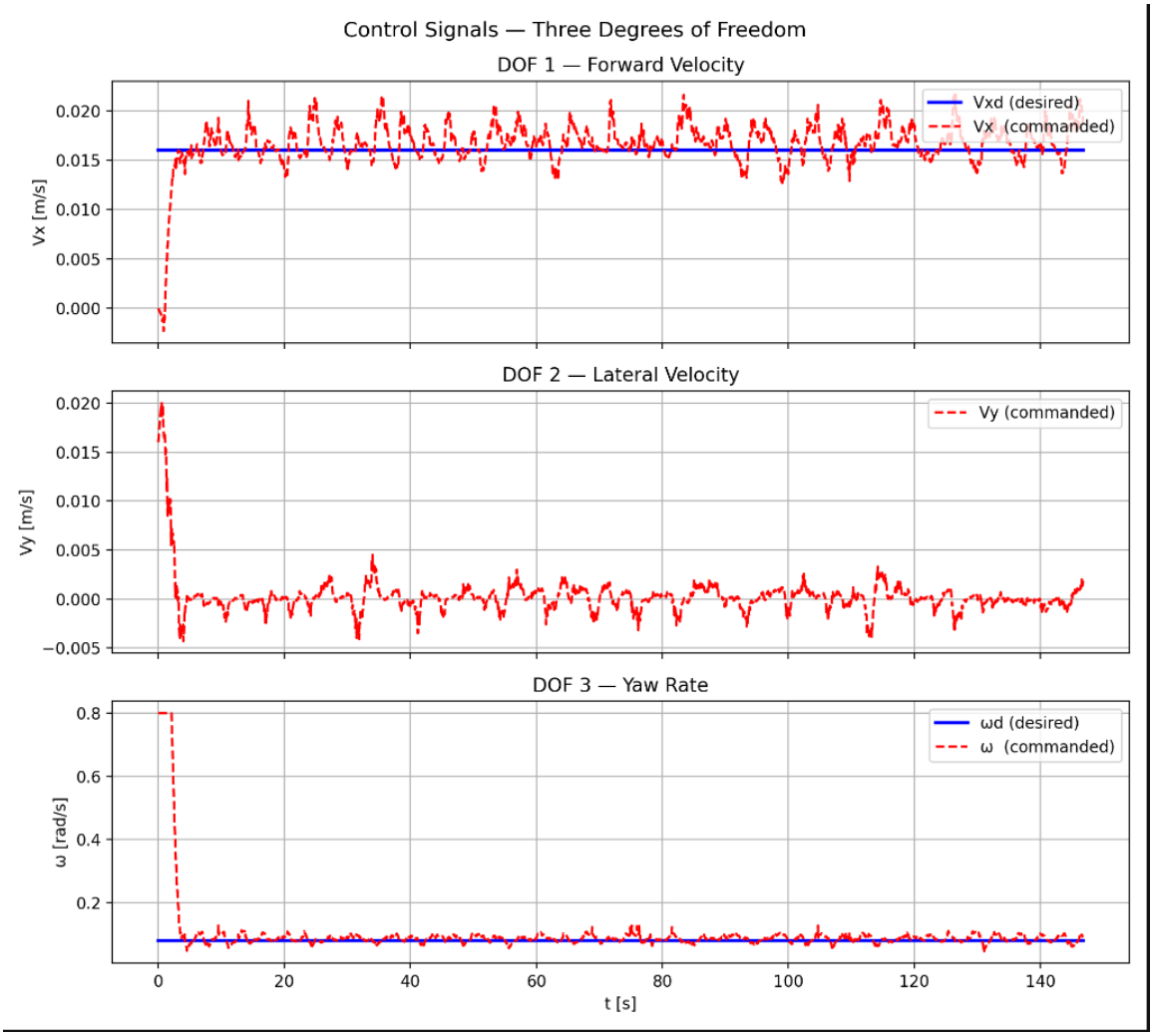


Figure 5.22: Hardware - Control signals for all three degrees of freedom. Top: DOF 1 forward velocity V_x vs desired V_{xd}. Middle: DOF 2 lateral velocity V_y (commanded). Bottom: DOF 3 yaw rate ω vs desired ω_d.

Adaptive gains. Figure 5.23 shows the three adaptive gains over time. Unlike simulation where gains settle to smooth values, the hardware gains show persistent high-frequency fluctuation due to real roller-ground disturbances. Despite this, all three gains remain within $[K_{\min}, K_{\max}]$ throughout the run, consistent with the Lyapunov stability bound of Equation (3.35).

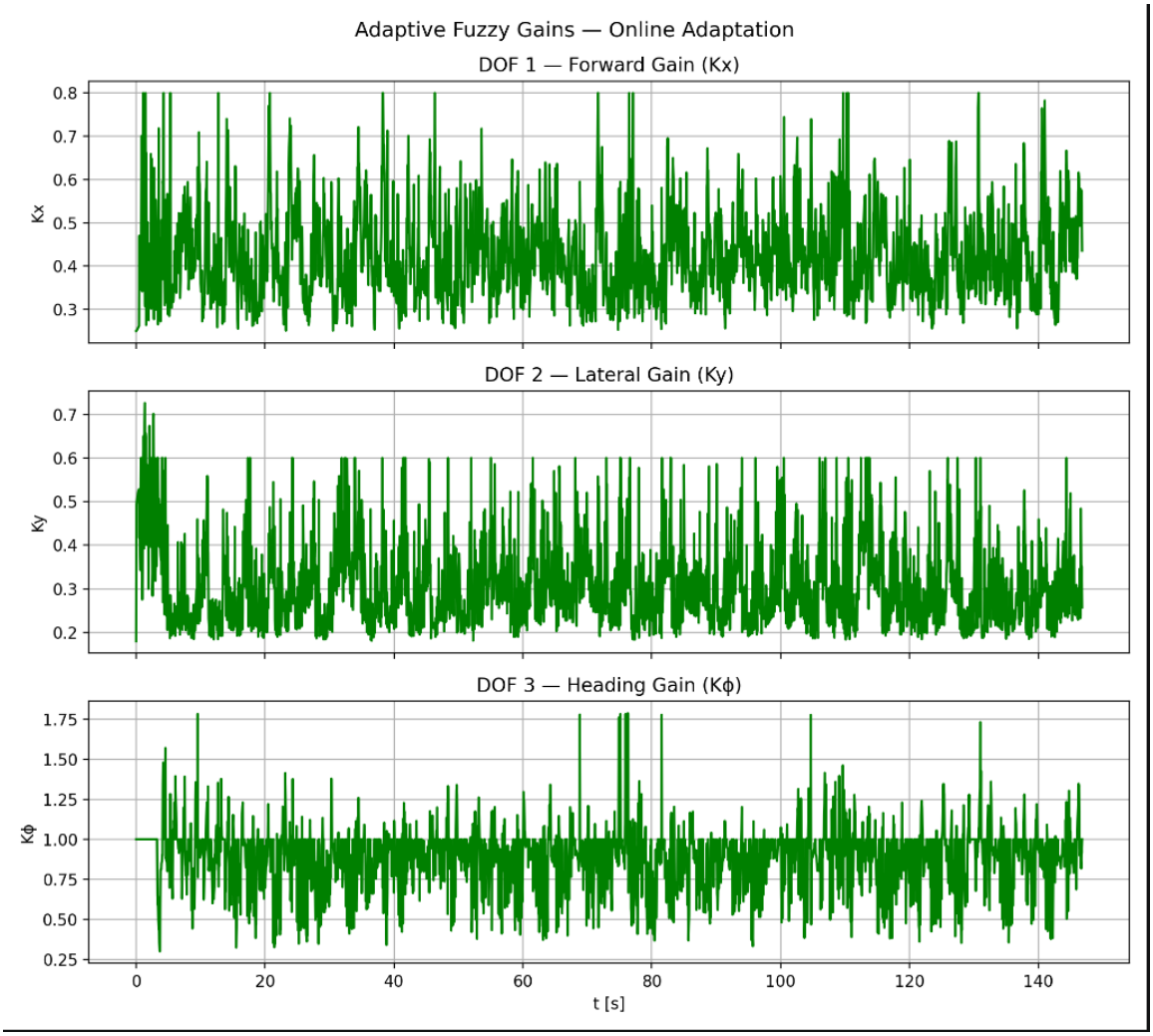


Figure 5.23: Hardware - Adaptive fuzzy gains for all three channels. Top: K_x (forward). Middle: K_y (lateral). Bottom: K_ϕ (heading). Gains remain bounded within $[K_{\min}, K_{\max}]$ throughout the run.

Body-frame errors. Figure 5.24 shows the three body-frame tracking errors. The heading error E_ϕ decays from its initial value to near zero within the first few seconds, reproducing the exponential convergence seen in simulation. The forward and lateral errors fluctuate around zero at a noise level approximately 2× higher than simulation, with no sustained drift or growing trend in any channel.

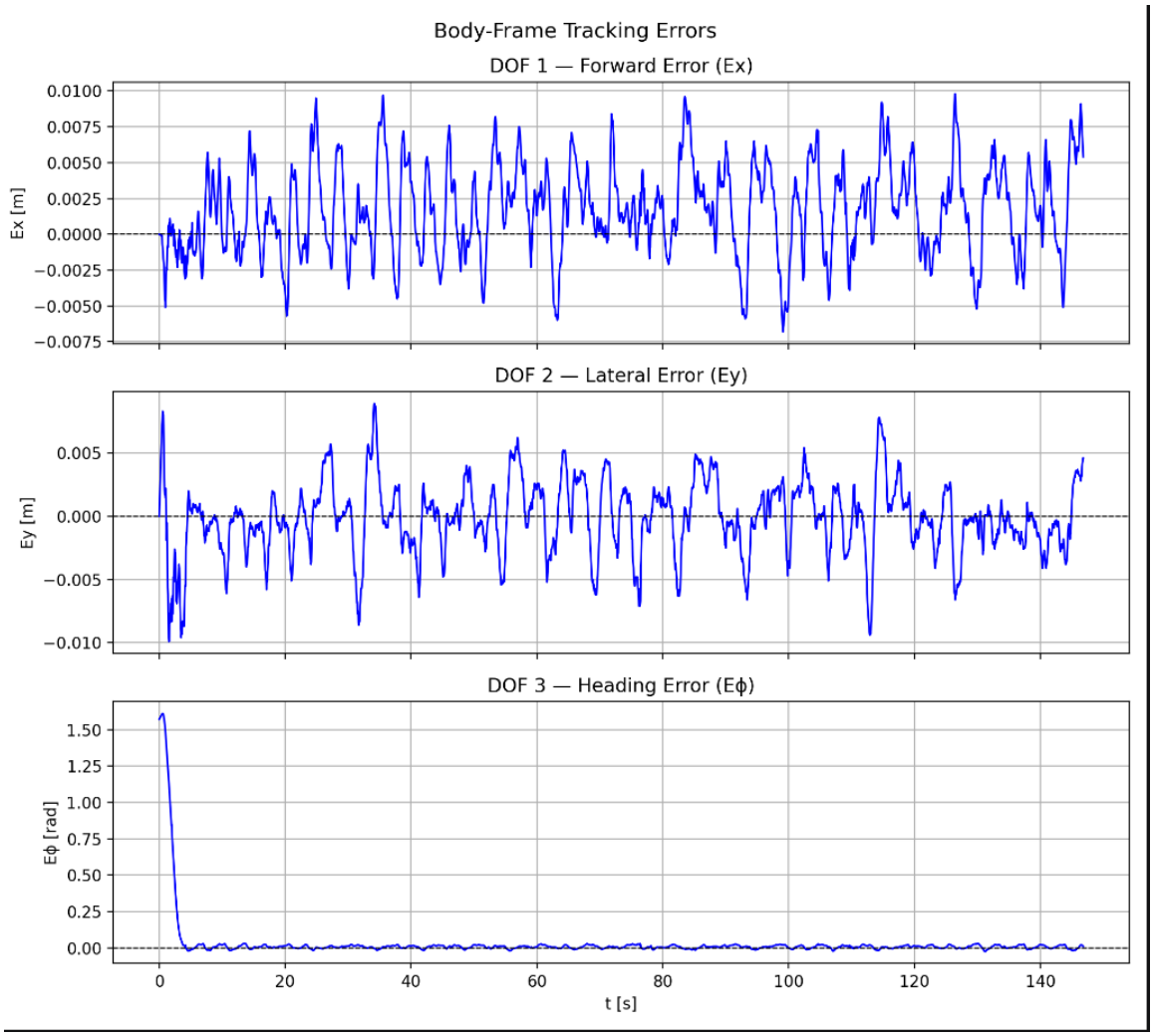


Figure 5.24: Hardware - Body-frame tracking errors. Top: E_x (forward). Middle: E_y (lateral). Bottom: E_ϕ (heading). The heading error decays from the initial value as in simulation. Position errors fluctuate around zero with no growing trend.

The hardware results confirm that the qualitative behaviour observed in simulation transfers directly to the physical platform. The controller converges onto the desired circle within the first quarter revolution on real hardware, reproducing the transient convergence seen in simulation. The steady-state RMSE of 0.0046 m is approximately 1.8× higher than the simulation value of 0.0026 m, which is consistent with the additional disturbance sources present on the physical platform: real roller vibration as each passive roller transitions between loaded and unloaded contact, velocity-dependent wheel slip on the laboratory floor surface and odometry drift accumulating over the 150 s run. The adaptive gains show persistent high-frequency fluctuation on hardware that is absent in simulation, reflecting these real contact disturbances being fed back through the error signal. Despite this, all three gains remain bounded within $[K_{\min}, K_{\max}]$ throughout the run and the body-frame errors show no sustained drift or growing trend in any channel, confirming that the Lyapunov stability guarantee holds under real operating conditions.

5.3 Summary of Results

Table 5.1 collects the key metrics from all experiments. The two failure mode runs give RMSE values of 0.36 m and 0.18 m respectively, illustrating the impact of incorrect parameterisation and timing mismatch. The correctly configured simulation run achieves $\text{RMSE}_{xy} = 0.0026$ m and the hardware run achieves $\text{RMSE}_{xy} = 0.0046$ m, approximately 1.8× higher due to real roller vibration, wheel slip and odometry drift.

Table 5.1: RMSE summary across all experiments.

Experiment	Description	RMSE (m)
Sim Failure Mode 1	Incorrect centre offset	0.36
Sim Failure Mode 2	Phase-lag (bandwidth exceeded)	0.18
Sim correct run	Correct configuration (simulation)	0.0026
Hardware run	Correct configuration (real robot)	0.0046

The two failure modes are not specific to the adaptive fuzzy controller. Any velocity-level kinematic controller on this platform would show the same problems under the same parameter choices. The trajectory initialisation error comes from a wrong centre offset. The phase-lag error comes from commanding a speed above the saturation limit. Both produce clear signatures in the error plots that make them straightforward to diagnose and fix.

The theoretical convergence and disturbance bounds derived in Section 3.3 are directly confirmed by the experimental results. The exponential convergence rate bounded below by $K_{\min}$ is visible in all runs: the heading error E_ϕ decays exponentially from its initial value to near zero within the first ten seconds in both the correctly configured simulation run and the outside spawn run, consistent with the bound $\|E(t)\| \leq \|E(0)\|e^{-K_{\min}t}$ from Equation (3.38). The outside spawn run provides the clearest illustration since $\|E(0)\|$ is large, making the exponential decay clearly visible in the body-frame error plots (Figure 5.18).

The disturbance bound $\delta/K_{\min}$ from Theorem 4.18 in Khalil (2002) is also consistent with the observed steady-state residuals. The simulation run achieves a steady-state RMSE of 0.0026 m, reflecting small unmodelled roller contact disturbances in Gazebo. The hardware run achieves 0.0046 m, approximately 1.8× higher, consistent with larger real disturbances from roller vibration, wheel slip and odometry drift on the physical platform. In both cases the residual remains small and bounded, confirming that the ultimate boundedness guarantee holds under real operating conditions.

Conclusion

6.1 Summary of Contributions

This thesis designed, implemented and tested an adaptive fuzzy gain-scheduled trajectory tracking controller for the Mecabot2. The controller works at the kinematic velocity level. That makes it directly compatible with the Mecabot2 hardware interface, which exposes only a velocity command topic and provides no access to motor torques.

The body-frame error dynamics for the mecanum platform were derived. A baseline proportional controller was shown to be globally asymptotically stable via a Lyapunov argument. The cross-coupling terms in the error dynamics cancel exactly because of the skew-symmetric structure of the equations. Three parallel Mamdani fuzzy systems then replaced the three fixed gains with adaptive gains computed online from the current error and its rate of change. The adaptive system inherits the same Lyapunov function. The convergence rate is bounded below by $K_{\min}$ and the residual error under bounded disturbances is bounded by $\delta/K_{\min}$.

The controller was implemented as a single ROS2 Python node at 20 Hz with full CSV logging of all signals. Every tunable quantity is set through a YAML file at launch. No recompilation is needed to change trajectory parameters or gain limits.

The same node ran without modification in both Gazebo simulation and real hardware experiments. In simulation the velocity commands were executed through the Gazebo mecanum drive plugin; on the physical robot the same commands were forwarded through the linorobot bringup chain and micro-ROS bridge to the embedded base controller. The control law, error formulation and gain-scheduling logic were identical in both cases. This confirms that the proposed controller operates at the ROS2 motion-control layer in a form that transfers directly from simulation to a real mecanum platform.

Gazebo simulation on a circular path with correct parameterisation gives an RMSE of 0.0026 m, with the initial transient decaying exponentially as predicted by the Lyapunov analysis. Two failure modes were documented with their diagnostic signatures and fixes. The trajectory initialisation error and the phase-lag instability both produce identifiable

patterns in the error time series that allow rapid diagnosis and correction. Hardware experiments on the physical Mecabot2 complement the simulation results.

6.2 Limitations

The simulation environment models each mecanum wheel as a single contact point with anisotropic friction. Real rollers vibrate as they hit the ground, slip in a velocity-dependent way and deform under load. None of that is fully captured in the simulation. The fuzzy parameters were tuned primarily in simulation. Real hardware conditions may require retuning of the membership function ranges and gain limits.

The controller works at the kinematic level. Inertial and Coriolis effects are treated as disturbances absorbed by feedback. That is justified below 0.8 m/s translation and 2.5 rad/s yaw. Above those limits the approximation breaks down and dynamic compensation becomes necessary.

The fuzzy rule base is symmetric and shared across all three channels. On real hardware the x, y and ϕ channels may behave differently depending on floor surface and load distribution. Separate rule bases or channel-specific normalisation scales could improve performance in those cases.

6.3 Future Work

The most immediate next step is a systematic hardware validation campaign covering multiple trajectories and floor surfaces. The ROS2 node transfers directly to the onboard Jetson Orin computer. The main task will be tuning the membership function ranges and gain limits for real contact conditions, using the CSV logging to compare the real and simulated signal profiles.

The second direction is testing on more trajectory shapes. Figure-eight paths and square paths with sharp corners would exercise the controller more aggressively and push the fuzzy gains toward their upper limit. These experiments would also reveal whether the symmetric rule base performs equally in all directions on real hardware.

The third direction is extension to the dynamic level. Adding a dynamic compensation layer would extend the operating range to higher speeds and reduce residual error under aggressive motion. The Lyapunov framework from this thesis is compatible with that extension since the stability proof structure is preserved when additional terms are included in the control law.

A fourth direction is the addition of a real-time visual marker for the desired trajectory point during both simulation and hardware experiments. In the current implementation the desired point $\mathbf{q}_d(t)$ moves along the reference circle invisibly. In simulation, adding a moving marker published as a RViz visualisation marker would allow real-time visual

verification that the robot is converging towards the correct point on the trajectory. On the physical Mecabot2, a similar effect could be achieved by publishing the desired pose as a ROS2 topic and overlaying it on the odometry trajectory in a post-processing visualisation tool. This improvement would be particularly useful during tuning and debugging, where it is important to distinguish between the robot tracking the correct phase of the trajectory and the robot simply moving in a circle with a phase offset, as documented in Failure Mode 2 (Section 5.1.2).

Bibliography

Cao, G., Zhao, X., Ye, C., Yu, S., Li, B., & Jiang, C. (2022). Fuzzy adaptive pid control method for multi-mecanum-wheeled mobile robot. *Journal of Mechanical Science and Technology*, *36*, 2019–2029. https://doi.org/10.1007/s12206-022-0337-x

Carvalho, J. P., Moreira, A. P., & Aguiar, A. P. (2024). Model predictive control for b-spline trajectory tracking in omnidirectional robots. *2024 7th Iberian Robotics Conference (ROBOT)*.

Chen, Y.-H. (2025). Nonlinear adaptive fuzzy hybrid sliding mode control design for trajectory tracking of autonomous mobile robots. *Mathematics*, *13*(8), 1329. https://doi.org/10.3390/math13081329

Ding, W., Zhang, J.-X., & Shi, P. (2024). Adaptive fuzzy control of wheeled mobile robots with prescribed trajectory tracking performance. *IEEE Transactions on Fuzzy Systems*, *32*(8).

Dixon, W. E., Dawson, D. M., Zergeroglu, E., & Behal, A. (2004). *Nonlinear control of wheeled mobile robots* (Vol. 262). Springer.

Fahmizal & Kuo, C.-H. (2016). Trajectory and heading tracking of a mecanum wheeled robot using fuzzy logic control. *2016 International Conference on Instrumentation, Control and Automation (ICA)*, 54–59. https://doi.org/10.1109/ICA.2016.7811475

Feng, X., & Wang, C. (2022). Adaptive neural network tracking control of an omnidirectional mobile robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, *237*(3). https://doi.org/10.1177/09596518221135904

Ha, V. T., Nguyen, V. H., & Le, A. T. (2019). Design of PID–backstepping neural networks controller for mecanum wheeled autonomous robot. *Proceedings of the International Conference on Engineering Research and Applications*, 362–372. https://doi.org/10.1007/978-3-030-37497-6_42

Huang, H., Liu, H., Li, M., & Ren, F. (2025). Robust trajectory tracking and obstacle avoidance control for omnidirectional mobile robots. *Asian Journal of Control*. https://doi.org/10.1002/asjc.3726

Ilon, B. E. (1975, April). *Wheels for a course stable self-propelling vehicle movable in any desired direction on the ground or some other base* (U.S. pat. No. 3876255) [U.S. Patent No. 3,876,255].

Khalil, H. K. (2002). *Nonlinear systems* (3rd). Prentice Hall.

Lamraoui, H. C., & Nemra, A. (2024). Trajectory tracking control for four mecanum wheels omnidirectional mobile robots based on active disturbances rejecter control. *2024 2nd International Conference on Electrical Engineering and Automatic Control (ICEEAC)*.

Li, Y., Qiu, L., Wang, Z., Zhou, J., & Guo, Y. (2023). Adaptive model predictive control for trajectory tracking of mecanum mobile robots. *2023 4th International Conference on Computer Engineering and Application (ICCEA)*. https://doi.org/10.1109/ICCEA58433.2023.10135332

Lima, G. d. S., Moreira, V. R. F., & Bessa, W. M. (2022). Accurate trajectory tracking control with adaptive neural networks for omnidirectional mobile robots subject to unmodelled dynamics. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, *44*(3), 1–15. https://doi.org/10.1007/s40430-022-03969-y

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, *7*(66), eabm6074. https://doi.org/10.1126/scirobotics.abm6074

Mai, T. A., Dang, T. S., Duong, D. T., Le, V. C., & Banerjee, S. (2021). A combined backstepping and adaptive fuzzy pid approach for trajectory tracking of autonomous mobile robots. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, *43*, 156. https://doi.org/10.1007/s40430-020-02767-8

Mamdani, E. H., & Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, *7*(1), 1–13. https://doi.org/10.1016/S0020-7373(75)80002-2

Moreno Caireta, I. (2019). *Model predictive control for a mecanum-wheeled robot in dynamical environments* [Master's thesis]. Universitat Politècnica de Catalunya.

Muir, P. F., & Neuman, C. P. (1987). Kinematic modeling of wheeled mobile robots. *Journal of Robotic Systems*, *4*(2), 281–340. https://doi.org/10.1002/rob.4620040209

Nguyen Minh, D., Do Quang, H., Dao Phuong, N., Ngo Manh, T., & Bui, D. N. (2023). An adaptive fuzzy dynamic surface control tracking algorithm for mecanum wheeled mobile robot. *International Journal of Mechanical Engineering and Robotics Research*, *12*(6), 354–361. https://doi.org/10.18178/ijmerr.12.6.354-361

Ortiz Hernández, J. C., & Rosas Almeida, D. I. (2024). Kinematic control in a four-wheeled mecanum mobile robot for trajectory tracking. *The Journal of Engineering*, *2024*(9), e70006. https://doi.org/10.1049/tje2.70006

Passino, K. M., & Yurkovich, S. (1998). *Fuzzy control*. Addison-Wesley.

Rojas, R., & Förster, A. G. (2006). Holonomic control of a robot with an omnidirectional drive [To appear in KI - Künstliche Intelligenz, 2006]. *KI – Künstliche Intelligenz*.

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). *Robotics: Modelling, planning and control*. Springer. https://doi.org/10.1007/978-1-84628-642-1

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots* (2nd). MIT Press.

Taheri, H., Qiao, B., & Ghaeminezhad, N. (2015). Kinematic model of a four mecanum wheeled mobile robot. *International Journal of Computer Applications*, *113*(3), 6–9. https://doi.org/10.5120/19830-1586

Wang, L.-X. (1994). *Adaptive fuzzy systems and control: Design and stability analysis*. Prentice Hall.

Wu, H.-M., & Karkoub, M. (2022). Frictional forces and torques compensation based cascaded sliding-mode tracking control for an uncertain omnidirectional mobile robot. *Measurement and Control*, *55*(1), 002029402210920. https://doi.org/10.1177/00202940221092033

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353. https://doi.org/10.1016/S0019-9958(65)90241-X

Zhao, T., Qin, P., & Zhong, Y. (2023). Trajectory tracking control method for omnidirectional mobile robot based on self-organizing fuzzy neural network and preview strategy. *Entropy*, *25*(2), 248. https://doi.org/10.3390/e25020248