UNIVERSITY OF CALIFORNIA
SANTA CRUZ

FEEDBACK CONTROL FOR AUTONOMOUS TRACTOR NAVIGATION

A dissertation submitted in partial satisfaction of the
requirements for the degree of

BACHELOR OF SCIENCE

in

ROBOTICS ENGINEERING

by

Hongtan Chen

June 2024

The Dissertation of Hongtan Chen
is approved:

Professor Dejan Milutinovic, Advisor

Professor Tae Myung Huh

Dean Alexander Wolf

Table of Contents

List of Figures

Abstract

Feedback Control for Autonomous Tractor Navigation

by

Hongtan Chen

This thesis explores the application of Nonlinear Control techniques and Proportional-Integral (PI) Controllers in autonomous vehicle navigation, specifically focusing on autonomous parking maneuvers for farming tractors within a simulated environment. It utilizes the Lyapunov stability theory to ensure stable and effective path tracking. The research demonstrates how advanced feedback control strategies can be optimized and applied in a virtual setting to allow precise vehicle control and trajectory management, paving the way for enhanced automation in agricultural practices.

To the Lord, my fortress

Whose steadfast love and strength have carried me through.

I dedicate this work in praise and thanks for the gifts bestowed upon me,

and for the enduring light on my path.

Acknowledgments

I extend my deepest gratitude to my advisor, Professor Milutinovic, whose expert guidance and continual support have been the cornerstone of my research. His insightful critiques and persistent patience not only pushed me to deepen my analytical reasoning but also encouraged me to achieve excellence in my thesis work. Professor Milutinovic excels in teaching several ECE courses, where his passion and experience in the subject matter have greatly influenced and inspired his students. His lectures are of high quality and deeply engaging. His ability to blend solid theoretical concepts with fascinating stories from his own experiences in the field makes his classes exceptionally informative and enjoyable. I am profoundly grateful for his mentorship, which has been invaluable in completing my degree and shaping my professional approach towards robotics and electrical engineering.

Further thanks to UC Santa Cruz and Baskin Engineering for providing me with 5 years of enriching academic environment. The curriculum and teaching staff have been tremendously supportive in equipping me with the necessary skills and development as an engineer. The challenging labs and assignments have propelled me to discover my potential and capabilities that I had previously underestimated. The department has nurtured a passion for continuous learning and innovation, and my heartfelt thanks go to everyone involved for their work and encouragement.

Finally, I would like to thank my parents for their unwavering support throughout my college journey. Their financial assistance and emotional backing have been

fundamental to my studies, allowing me to focus fully on my academic goals. I am deeply grateful for their sacrifices and constant love, which have made all the difference in my educational and personal development.

Chapter 1

Introduction

Autonomous vehicle technology has been revolutionizing various industries by enhancing safety, efficiency, precision, and convenience in operation in the recent years. In agriculture, vehicles such as autonomous farming tractors can significantly improve productivity and precision, and therefore reduce the need for human labor and minimize errors. This senior thesis aims to contribute to the field of autonomous navigation by integrating feedback control techniques to achieve reliable and efficient autonomous parking maneuvers for farming tractors.

A variety of strategies have been researched and developed to address autonomous vehicle navigation in complex real-world environments. Autonomous navigation presents unique challenges, such as dynamic obstacle avoidance, precision in path following, and the need for robust operation under diverse environmental conditions. The existing literature reveals a spectrum of methods, from traditional algorithms such as the Dijkstra algorithm [6] to modern machine learning approaches like reinforcement

learning [4], each suited to different aspects of these challenges.

One classic navigation algorithm is the Pure Pursuit method [8], which is a path-following algorithm that controls the vehicle's angular velocity and directs the vehicle toward the instantaneous target position while maintaining a constant forward velocity. It has a respectable history of use in various autonomous navigation applications and has demonstrated reliability. In the context of navigation perception, Simultaneous Localization and Mapping (SLAM) [14], while it's not a navigation algorithm, it enables vehicles to determine their real-time location in environments where GPS signals are unavailable or unreachable. This capability helps the navigation algorithms use LiDAR sensors to create detailed maps of the surroundings so that the vehicle can navigate without reliance on GPS in such as indoor environments or urban areas with obstructed satellite signals.

Some other modern methods that have been employed in obstacle avoidance and navigation tasks include Reinforcement Learning (RL) and Deep Learning (DL) [3]. Autonomous navigation can be achieved by training a neural network agent to map its estimated state to acceleration and steering commands while considering obstacles. This method has been successfully applied to full-size research vehicles and has been proven practical.

Additionally, researchers have conducted comparative studies among path-tracking controllers for navigating autonomous vehicles [5]. Path-tracking controllers like the Stanley method, PID control, linear quadratic regulator, sliding mode control, and model predictive control have been explored and compared in researching four-

wheel steering on low-friction roads. Researchers have also looked into steering and yaw moment control maneuvers and the effectiveness of model predictive controller and linear quadratic controller for stability control of the vehicle's lateral position and yaw angle [13]. These two controllers were evaluated based on the tracking performance at different speeds and road surface conditions, and the model predictive control is shown more favorable for maintaining vehicle stability and trajectory over the linear quadratic control.

Nonlinear control techniques have also been studied in navigating mobile robots [7], particularly nonholonomic systems where constraints are velocity-based rather than position-based, where pure state feedback stabilization around a given terminal configuration is impossible. Nonlinear feedback control methods have been practically demonstrated to achieve precise control of the robot's position and orientation in Cartesian space. Similarly, nonlinear feedback control was also applied to the dynamic a mobile robot's dynamic model [10] in order to address the nonlinearities inherent in the robot's dynamics to stabilize its trajectory and improve navigation in complex environments.

This thesis focuses on the integration of nonlinear control techniques and PI controllers within a simulated environment to explore their effectiveness in real-time autonomous navigation. The subsequent chapters will delve into the details of the methodology that we used to achieve the autonomous parking maneuver for our farming tractor. Chapter 2 explores the theoretical foundation of the Lyapunov stability theory applied to ensure that the control strategy leads to a stable and convergent navigation

path for our autonomous tractor. Chapter 3 examines the Proportional-Integral (PI) controllers used to translate the desired velocities and turning rates from the nonlinear controller into actual motor commands for the tractor. This chapter includes the selection and tuning of PI controller parameters, the analysis of their impact through root locus plots, and their effectiveness in managing the actual driving dynamics of the vehicle. Chapter 4 discusses the detailed implementation and tuning of the nonlinear controller, which calculates the dynamic control inputs (velocity and turning rate) based on the current state of the tractor relative to the target and adjusts the tractor's trajectory toward the target position and orientation.

Chapter 2

Lyapunov Function

The Lyapunov function concept, is a tool for analysis of the stability of equilibrium points in dynamical systems [9]. In a dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^N$, an equilibrium, i.e., fixed point $\mathbf{x}^* \in \mathbb{R}^N$ is the state space point at which $f(\mathbf{x}^*) = \mathbf{0}$. In tha analysis of the fixed point stability, a Lyapunov function $V(\mathbf{x})$ (when it exists) has two main properties:

1. $V(\mathbf{x}) > 0$ for all $\mathbf{x} \neq \mathbf{x}^*$, and $V(\mathbf{x}^*) = 0$ (function V is positive definite.)

2. $\dot{V} < 0$ for all $\mathbf{x} \neq \mathbf{x}^*$ (the derivative of V along the system's trajectories is negative.)

The main result of stability analysis is that when a function $V(\mathbf{x})$ exists, the fixed point $\mathbf{x}^*$ is globally asymptotically stable for all initial conditions, which means that as $t \to \infty$ we have $\mathbf{x}(t) \to \mathbf{x}^*$. The Lyapunov function concept has been also used to prove that a dynamical system does not have closed orbits.

Linear systems: To illustrate the concept let us consider a linear dynamical

system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is a square $N \times N$ matrix since $\dim(\dot{x}) = \dim(x) = N$. Let us define a Lyapunov function as $V(\mathbf{x}) = \mathbf{x}^T\mathbf{x} = \sum_i x_i^2 > 0$, for all $\mathbf{x} \neq \mathbf{0}$. The system fixed point is $\mathbf{x} = \mathbf{0}$, which is stable if we can show that:

$$\dot{V} = \dot{\mathbf{x}}^T\mathbf{x} + \mathbf{x}^T\dot{\mathbf{x}} = \mathbf{x}^T\mathbf{A}^T\mathbf{x} + \mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^\mathbf{T}(\mathbf{A}^\mathbf{T} + \mathbf{A})\,\mathbf{x} < \mathbf{0},\ \textit{for all}\ \mathbf{x} \neq \mathbf{0} \tag{2.1}$$

Obviously, this is correct for all $\mathbf{x} \neq \mathbf{0}$, if and only if $\mathbf{A}^T + \mathbf{A}$ is a negative definite matrix. Note that the matrix $\mathbf{A}^T + \mathbf{A}$ is not only a square but also a symmetric matrix.

Let us consider a square matrix $\mathbf{M} = \frac{\mathbf{A}+\mathbf{A}^T}{2}$ and algebraic equations

$$(\mathbf{M} - \mu_i\mathbf{I})\mathbf{v}_i = \mathbf{0}, \quad i = 1, ...N \tag{2.2}$$

where $\mathbf{I}$ is the unity matrix, μ_i, $i = 1, 2...N$ are the so-called eigenvalues and $\mathbf{v}_i$ are their corresponding eigenvectors. In general, the eigenvalues μ_i are complex numbers and have well defined smallest m and largest M real parts, i.e.,

$$m = \min_i\{\mathbf{Re}(\mu_1), ...\mathbf{Re}(\mu_N)\}, \quad M = \max_i\{\mathbf{Re}(\mu_1), ...\mathbf{Re}(\mu_N)\}, \tag{2.3}$$

According to the Bendixson's inequality [11] them all real parts of matrix $\mathbf{A}$ eigenvalues λ_i, $i = 1, 2, ...N$ satisfy

$$m \leq \mathbf{Re}(\lambda_i) \leq M,\ i = 1, 2, ...N \tag{2.4}$$

Stating that the matrix $\mathbf{A} + \mathbf{A}^T$ is negative definite is equivalent to state that

$\mathbf{M} = \frac{\mathbf{A}+\mathbf{A}^T}{2}$ is negative definite, i.e., $\mathbf{x}^T\mathbf{M}\mathbf{x} < 0$ for all non-zero $\mathbf{x} \in \mathbb{R}^n$. Also according to the fundamental properties of negative definite matrices, all of their eigenvalues have negative real parts, i.e.,

$$\mathbf{x}^T\mathbf{M}\mathbf{x} < 0 \Rightarrow \mathbf{Re}(\lambda_i) \leq M < 0 \tag{2.5}$$

In other words the matrix $\mathbf{A} + \mathbf{A}^T$ is negative definite if all of the matrix $\mathbf{A}$ eigenvalues are negative and we can conclude that

$$\dot{V}(\mathbf{x}) < 0 \Rightarrow \text{all eigenvalues of } \mathbf{A} \text{ have negative real parts.} \tag{2.6}$$

which is the property of stable system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$. We can also show that

$$\text{all eigenvalues of } \mathbf{A} \text{ have negative real parts} \Rightarrow \dot{V}(\mathbf{x}) < 0 \tag{2.7}$$

Note that if $\mathbf{A}$ has all eigenvalues with negative real parts, then $\mathbf{A}$ is negative-definite, i.e.

$$\mathbf{x}^\mathbf{T}\mathbf{A}\mathbf{x} = c < 0 \tag{2.8}$$

since c is a scalar value and we know that $c^T = c$, therefore,

$$(\mathbf{x}^T\mathbf{A}\mathbf{x})^T = c^T \Rightarrow \mathbf{x}^T\mathbf{A}^T\mathbf{x} = c < 0 \tag{2.9}$$

In other words, if $\mathbf{A}$ is a negative-definite matrix, then $\mathbf{A}^T$ is also a negative-definite

matrix. Therefore, we can conclude:

$$\dot{V}(\mathbf{x}) = \mathbf{x}^T(\mathbf{A} + \mathbf{A}^T)\mathbf{x} = 2\mathbf{x}^T\mathbf{A}\mathbf{x} = 2c < 0 \tag{2.10}$$

From this and (2.7), we can show that:

$$\text{All eigenvalues of } \mathbf{A} \text{ have negative real parts} \Leftrightarrow \dot{V}(\mathbf{x}) \tag{2.11}$$

Which is one of the best-known results for the stability of linear systems stating that a linear system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ is stable if and only if all eigenvalues of matrix $\mathbf{A}$ have negative real parts.

Linear system example 1: Let us consider a simple 2-dimensional system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} \underbrace{\begin{bmatrix} -2 & 0 \\ 0 & -3 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{2.12}$$

We can find that the eigenvalues of $\mathbf{A}$ are $\lambda_1 = -2$ and $\lambda_2 = -3$. Since both of them are negative, we can conclude that the system is stable.

To analyze the stability, we can also use the Lyapunov function $V(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$, which is a scalar function representing the "energy" of the system at state $\mathbf{x}$. The

Lyapunov function is:

$$V(\mathbf{x}) = \mathbf{x}^T\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 + x_2^2 \tag{2.13}$$

The derivative of V with respect to time along the trajectories of the system is:

$$\dot{V} = \frac{d}{dt}(x_1^2 + x_2^2) = 2x_1\dot{x}_1 + 2x_2\dot{x}_2 \tag{2.14}$$

Substituting $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ into $\dot{V}$, i.e., $\dot{x}_1 = -2x_1$ and $\dot{x}_2 = -3x_2$, we get:

$$\dot{V} = 2x_1(-2x_1) + 2x_2(-3x_2) = -4x_1^2 - 6x_2^2 < 0 \tag{2.15}$$

This Lyapunov function $V(x)$ is positive-definite since $V(x) > 0$ for all $x \neq 0$ and $V(0) = 0$. The derivative $\dot{V}$ is negative-definite since $\dot{V} < 0$ for all $x \neq 0$. The conditions indicate that all trajectories will converge to the origin as $t \to \infty$ for this system. Hence the system is stable.

In this chapter, we've presented the foundational principles of Lyapunov stability theory and its application to both linear and nonlinear dynamical systems. By employing Lyapunov functions, we demonstrated a systematic approach to proving the stability of equilibrium points in such systems. For linear systems, the use of a quadratic Lyapunov function illustrated how the stability of a system could be conclusively determined by examining the eigenvalues of the system matrix. The negative definiteness of the

matrix derived from the Lyapunov function's derivative provided a robust criterion for stability, ensuring that all system trajectories converge to the equilibrium point as time progresses. This theoretical groundwork is essential for the subsequent chapters, where these principles are applied to develop and analyze control strategies for the tractor.

Chapter 3

PI Controller for Turning Rate and Velocity

The Proportional-Integral (PI) controllers are implemented to provide a desired velocity and turning rate for actual vehicle kinematics. The design and tuning of these controllers are to dynamically adjust the turning rates of the vehicle wheels to achieve the vehicle's velocity and turning rate required by the nonlinear navigation controller.

The velocity control loop is designed to regulate the speed of the tractor by adjusting the motor driving inputs based on the difference between a desired reference velocity v_{Ref} and measured velocity v_{Meas}. The velocity feedback control loop diagram is depicted in Fig. 3.1.

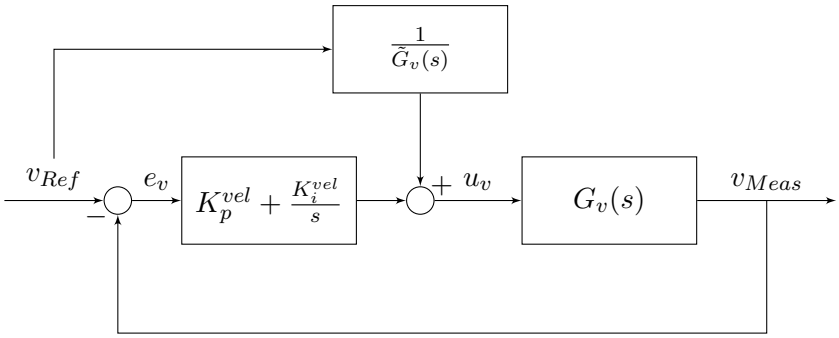

Figure 3.1: Proportional-Integral (PI) Controller for Velocity

The PI controller output consists of the sum of two components. One of them is proportional to the velocity error ($v_{Ref} - v_{Meas}$) and it has the gain K_p^{vel}. The other is proportional to the integral of the velocity error, i.e., $\int_0^t (v_{Ref} - v_{Meas})dt$, and it has the gain K_i^{vel}. In our feedback control loop, the integral component is introduced with an idea to remove any residual steady-state error in tracking a constant velocity reference. This will be confirmed once we the vehicle's velocity model $G_v(s)$

In our feedback control design we also have a feedforward compensation represented by the block $\frac{1}{\tilde{G}_v(s)}$. The block generates the output which matches the value of signal u_v required for a desired velocity v_{Mes}, for this reason the feedforward compensation is the inverse of the plant's transfer function $G_v(s)$ from u_v to v_{Meas}. However, the match is usually not exact so we can say $\tilde{G}_v(s) \approx G_v(s)$. Overall the controller output signal u_v is :

$$u_v(t) = \underbrace{K_p^{vel} \cdot e_v(t) + K_i^{vel} \int e_v(t)dt}_{PIcontroller} + Feedforward \tag{3.1}$$

or in the Laplace domain

$$u_v(s) = \underbrace{\left(K_p^{vel} + \frac{K_i^{vel}}{s}\right)}_{C_v(s)} e_v(s) + \underbrace{\frac{1}{\tilde{G}_v(s)} V_{ref}(s)}_{Feedforward} \tag{3.2}$$

where the error $e_v(s) = v_{Ref}(s) - v_{Meas}(s)$ and $C_v(s)$ denotes the transfer function of the PI controller.

The feedback control loop diagram for the turning rate is depicted in Fig. 3.2 and it has the same structure as the one for the velocity.

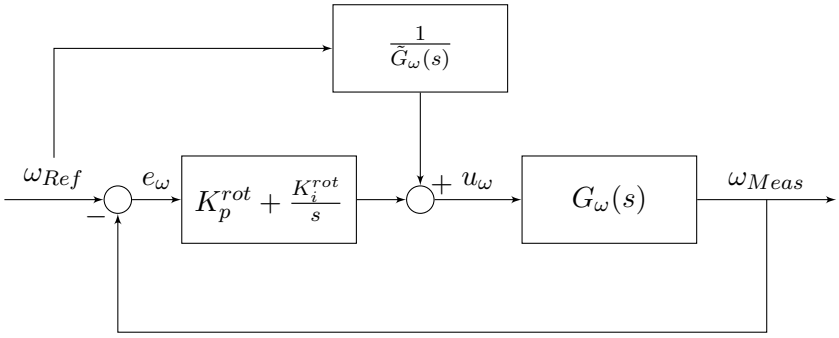

Figure 3.2: Proportional-Integral Controller for Turning Rate

In this case the proportional and integral gains are K_p^{rot} and K_i^{rot} and in the Laplace domain the controller is

$$u_\omega(s) = \underbrace{\left(K_p^{rot} + \frac{K_i^{rot}}{s}\right)}_{C_\omega(s)} e_v(s) + \underbrace{\frac{1}{\tilde{G}_\omega(s)} \omega_{ref}(s)}_{Feedforward} \tag{3.3}$$

where $e_\omega(s) = \omega_{Ref} - \omega_{Meas}$ is the reference turning rate ω_{Ref} tracking error and $G_\omega(s)$ is the transfer function from the control output u_ω to the measured turning rate ω_{Meas}.

Thus, the feedforward compensation block is $\frac{1}{\tilde{G}_\omega(s)}$, where $\tilde{G}_\omega(s) \approx G_\omega(s)$. In the last expression $C_\omega(s)$ denotes the transfer function of the PI controller.

The effectiveness of both PI controllers, $C_v(s)$ and $C_\omega(s)$, are highly dependent on their correct tuning, i.e., values of the proportional and integral gains. Proper tuning ensures that the vehicle can smoothly follow its trajectory with minimal overshoot and oscillations, which must be adjusted based on the analysis of the transfer function, specific dynamics of the tractor, and its environment. Before we get into the details of the transfer functions $G_v(s)$ and G_ω we will find the transfer function of our two feedback controllers including feedforward compensation. Since both controllers have the same structure in sequel we will use $C(s)$ for both $C_v(s)$ and $C_\omega(s)$, and $G(s)$ for both $G_v(s)$ and $G_\omega(s)$. The references v_{Ref} and ω_{Ref} will be denoted by r and the measured outputs $v_{Meas}(s)$ and $\omega_{Meas}(s)$ will be denoted by y. Our first goal here is to find the transfer function $\frac{y(s)}{r(s)}$, then we will find the transfer function describing the error dunamics $e(s) = r(s) - y(s)$. To achieve that we will start from the following relation depicted in in Fig. 3.1 as well as in Fig. 3.2

$$y(s) = G(s)\left(C(s)(r(s) - y(s)) + \frac{1}{\tilde{G}(s)}r(s)\right) \tag{3.4}$$

This relation can be rewritten as

$$y(s)(1 + G(s)C(s)) = G(s)C(s)r(s) + \frac{G(s)}{\tilde{G}(s)}r(s) \tag{3.5}$$

and we can conclude that

$$y(s) = \frac{G(s)C(s) + G(s)/\tilde{G}(s)}{1 + G(s)C(s)} r(s) \tag{3.6}$$

From this, the error dynamics is described as

$$e(s) = r(s) - y(s) = \frac{1 + G(s)C(s) - G(s)C(s) - G(s)/\tilde{G}(s)}{1 + G(s)C(s)} r(s) \tag{3.7}$$

i.e.,

$$e(s) = \frac{1 - G(s)/\tilde{G}(s)}{1 + G(s)C(s)} r(s) \tag{3.8}$$

which reflect the fact that the only feedback loop of the block diagram is the one with the loop gain $L(s) = G(s)C(s)$ and that the stability of that loop can be determined from the roots of

$$1 + G(s)C(s) = 0 \tag{3.9}$$

It is reasonable to assume that if both $G(s)$ and $G\tilde{(}s)$ are stable that the stability analysis of the loop is sufficient to determine the stability of the overall system. This should be checked for specific transfer function $G(s)$ and $\tilde{G}(s)$.

We've identified that the system behaves with a static gain. This simplifies the control dynamics and makes it easier to understand and tune the controllers based on the system's response. A static gain essentially means that the system's response can be

predominantly characterized by a constant gain at steady state.

To effectively tune the PI controllers and provide the feedforward mechanism, it is essential to determine the plant constant G, which characterizes the linear dynamic behavior of the tractor. For the velocity control system, the static gain G_v represents how effectively a change in the control signal uV affects the actual velocity $uMes$ of the tractor. Similarly, for turning rate control, the static gain G_ω quantifies the responsiveness of the turning rate $wMes$ to changes in the control signal $uOmega$

The plant constant G can be experimentally obtained by analyzing the system's response to a known input signal. One common method involves applying a square wave input and measuring the corresponding output.

To determine the plant constants G_v (velocity) and G_ω (turning rate), we employed an experimental approach involving the application of a step input signal to the system and measuring the resulting steady-state output. The plant constants are derived as the ratio of the change in the output to the change in the input.

We began by modifying the control inputs to apply a step input signal for a fixed duration. This was done by setting the motor input signals uV and $uOmega$ to known values and observing the corresponding outputs. The output responses $uMes$ (measured velocity) and $wMes$ (measured turning rate) were then recorded during the application of the step input. The experimental setup was implemented as illustrated in the following algorithm.

Algorithm 1 Determine Plant Constants G_v and G_w

Initialize global variables for PI controllers
while simulation is running **do**
    Overwrite PI controller outputs for measurement
    if Providing turning rate step input **then**
        $uV \leftarrow 0$
        $uOmega \leftarrow 1$ ▷ Turning rate step input
    else if Providing velocity step input **then**
        $uV \leftarrow 1$ ▷ Velocity step input
        $uOmega \leftarrow 0$
    end if
    Send left and right motor velocities
    sim.setJointTargetVelocity($motorLeft1, uV - uOmega$)
    sim.setJointTargetVelocity($motorLeft2, uV - uOmega$)
    sim.setJointTargetVelocity($motorRight1, uV + uOmega$)
    sim.setJointTargetVelocity($motorRight2, uV + uOmega$)
    Print the measured output for analysis
    print($uMes, wMes$)
end while
Compute plant constants G_v and G_w
$G_v \leftarrow \frac{uMes}{uV}$ ▷ Calculate velocity plant constant
$G_w \leftarrow \frac{wMes}{uOmega}$ ▷ Calculate turning rate plant constant

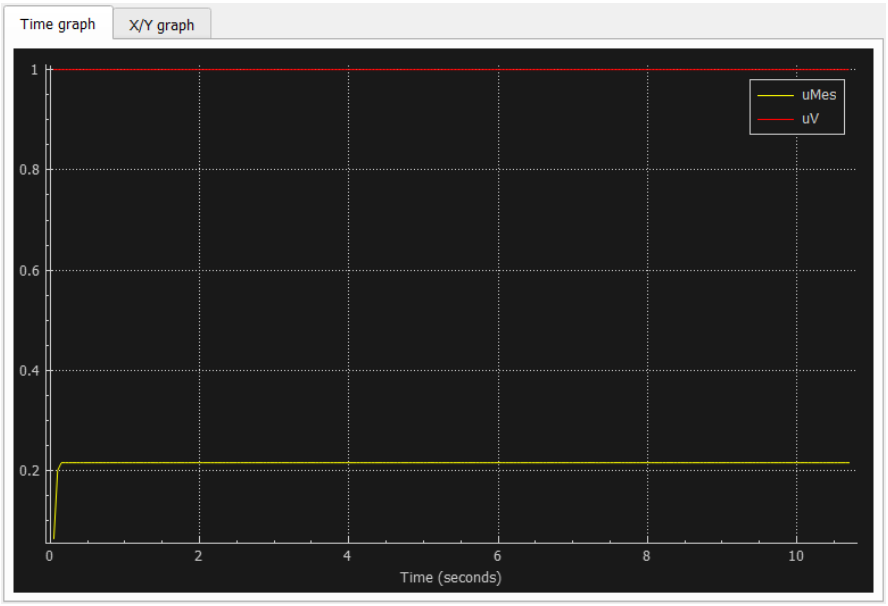

Figure 3.3: Step Response for Plant G_v

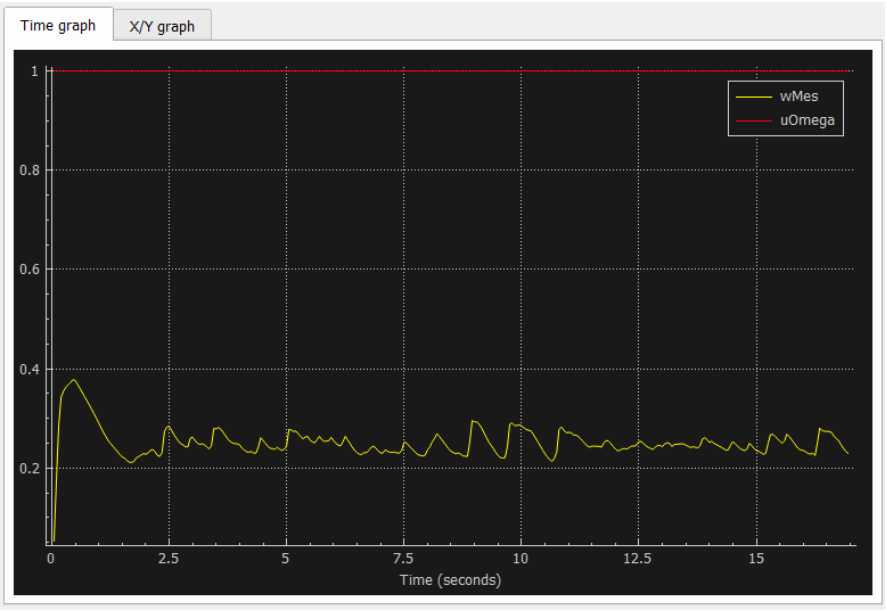

Figure 3.4: Step Response for Plant G_ω

The recorded data was analyzed to determine the plant constants. We identified the moments in Fig. 3.3 when the velocity input changed (step input) and measured the corresponding change in the velocity output. Similarly, Fig. 3.4 shows the change in the turning rate output when the turning rate input changes. The plant constants G_v and G_ω were calculated as the ratios of the change in output to the change in input.

For instance, if uV changes from 0 to 0.2 and $uMes$ changes from 0 to 0.1, then:

$$G_v = \frac{\Delta\text{Output}}{\Delta\text{Input}} = \frac{0.1}{0.2} = 0.5 \tag{3.10}$$

Similarly, if $uOmega$ changes from 0 to 0.01 and $wMes$ changes from 0 to 0.005, then:

$$G_\omega = \frac{\Delta\text{Output}}{\Delta\text{Input}} = \frac{0.005}{0.01} = 0.5 \tag{3.11}$$

From the experimental data, we observed that the ratio for turning rate fluctuated between 0.221 and 0.256, averaging approximately 0.238,

$$G_\omega = 0.238 \tag{3.12}$$

This value indicates that a unit step change in $uOmega$ will result in a change of about 0.238 units in the measured turning rate $wMes$, assuming other dynamics are negligible and the system is at steady-state.

The ratio for velocity was around 0.2159.

$$G_v = 0.2159 \tag{3.13}$$

This value means that for a unit step increase in uV, the velocity $uMes$ increases by approximately 0.2159 units under steady-state conditions.

To carefully analyze the stability of our PI controller systems, we need to calculate our loop transfer functions. Since both PI controllers are computationally equivalent with different gain variables, we'll analyze using general terms representing the gains.

The PI controller transfer function is defined as:

$$C(s) = K_p + \frac{K_i}{s} \tag{3.14}$$

The Plant transfer function which represents the linear dynamic behavior of

the tractor is defined as:

$$Plant(s) = G(s) \tag{3.15}$$

Therefore, we can derive the open-loop transfer function (loop gain):

$$\begin{aligned} OLTF(s) &= C(s) \times G(s) \\ &= \left(K_p + \frac{K_i}{s}\right) \times G(s) \\ &= K_p \left(s + \frac{K_i}{K_p}\right) \frac{G(s)}{s} \end{aligned} \tag{3.16}$$

The closed-loop transfer function can be derived as:

$$\begin{aligned} CLTF(s) &= \frac{OLTF(s)}{1 + OLTF(s)} \\ &= \frac{G(s)K_p + \frac{G(s)K_i}{s}}{1 + G(s)K_p + \frac{G(s)K_i}{s}} \\ &= \frac{sG(s)K_p + G(s)K_i}{s + sG(s)K_p + G(s)K_i} \\ &= \frac{sG(s)K_p + G(s)K_i}{s[1 + G(s)K_p] + G(s)K_i} \end{aligned} \tag{3.17}$$

The roots of the denominator of the closed-loop transfer function can be obtained from

$$s + sG(s)K_p + G(s)K_i = 0 \tag{3.18}$$

which yields

$$s = \frac{-G(s)K_i}{1 + G(s)K_p} \tag{3.19}$$

Given that K_p, K_i, and $G(s)$ are positive, the pole is negative, indicating that the system is stable since the pole lies in the left-half of the s-plane. This provides a theoretical confirmation of stability under positive gains and system response parameters.

To apply Root Locus analysis, we need to understand how the poles of the closed-loop system move in the s-plane as we vary the gain. Since we know that our characteristic equation derived from the closed-loop transfer function with the plant transfer function $G(s)$ being constant is:

$$s + sGK_p + GK_i = 0 \tag{3.20}$$

The poles of the system are:

$$s = \frac{-GK_i}{1 + GK_p} \tag{3.21}$$

The poles of the velocity controller are:

$$s = \frac{-G_v K_i}{1 + G_v K_p} = \frac{-0.2159 K_i}{1 + 0.2159 K_p} \tag{3.22}$$

The poles of the turning rate controller are:

$$s = \frac{-G_\omega K_i}{1 + G_\omega K_p} = \frac{-0.238 K_i}{1 + 0.238 K_p} \tag{3.23}$$

For the system to be stable, the poles must be in the left half of the s-plane:

$$\mathrm{Re}(s) < 0 \tag{3.24}$$

In digital control systems, sampling rate is a crucial factor because it dictates how often the controller reads sensors and updates actuators. The sampling rate must be adequately high relative to the dynamics of the system being controlled to ensure stability and performance. This principle is often guided by the Nyquist-Shannon sampling theorem [12], which states that to accurately capture a signal, the sampling frequency must be at least twice the highest frequency present in the signal (the Nyquist rate). For our PI controller systems, the sampling rate should be much higher, typically 6 to 10 times higher than the bandwidth of the system's power response (the frequency at which the system's response starts to significantly diminish). The higher sampling rate helps in capturing the dynamics of the system more accurately.

Our sampling time (T_s) is 50 ms,

$$f_s = \frac{1}{T_s} = \frac{1}{0.05} = 20 \text{ Hz} \tag{3.25}$$

and since we want our sampling rate to be at least 6 times faster, the sampling frequency should be:

$$f_d \geq 6 \times f_s = 6 \times 20 = 120 \text{ Hz} \tag{3.26}$$

From the loop gain in Eq. 3.16, we can choose $\frac{K_i}{K_p}$ while K_p serves as positive K from the Root Locus method. Since the sketch of the Root Locus will show that there is only one single branch to the Root Locus departing ($K = 0$) from the pole at the origin and arriving ($K = \infty$) to the zero at $\frac{K_i}{K_p}$, it is important to show that the system pole is about 10 to 6 smaller than f_s from Eq. 3.25. The value of $\frac{K_i}{K_p} = 3$ is a safe choice

since $3 < \frac{20}{6}$.

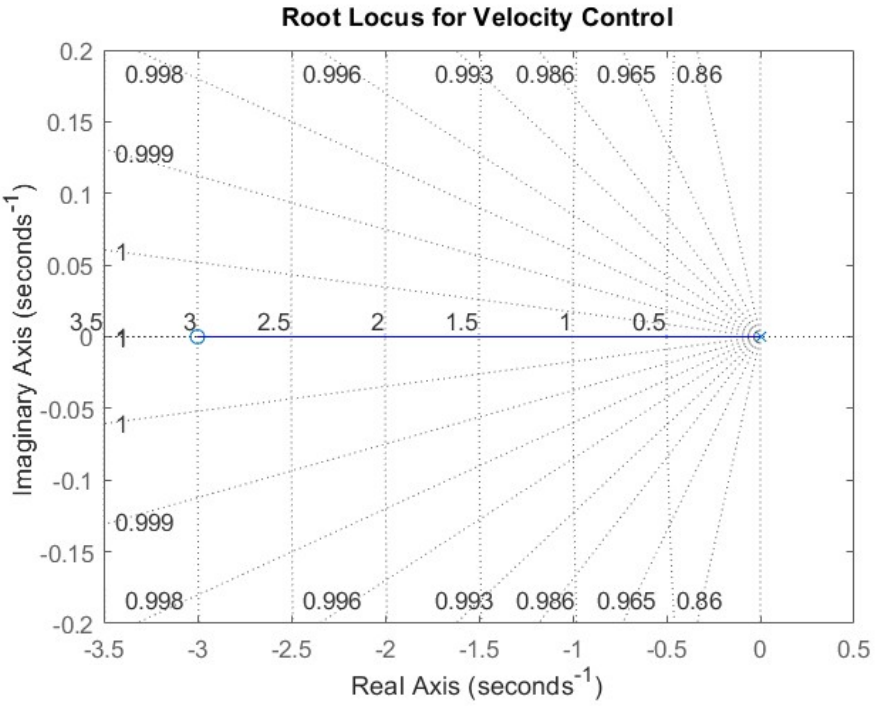

Figure 3.5: Root Locus for Velocity Control: The plot depicts the path of closed-loop poles for velocity control with increasing feedback gain K. It shows the poles starting from the origin and moving towards the zero defined by $\frac{K_i}{K_p}$.

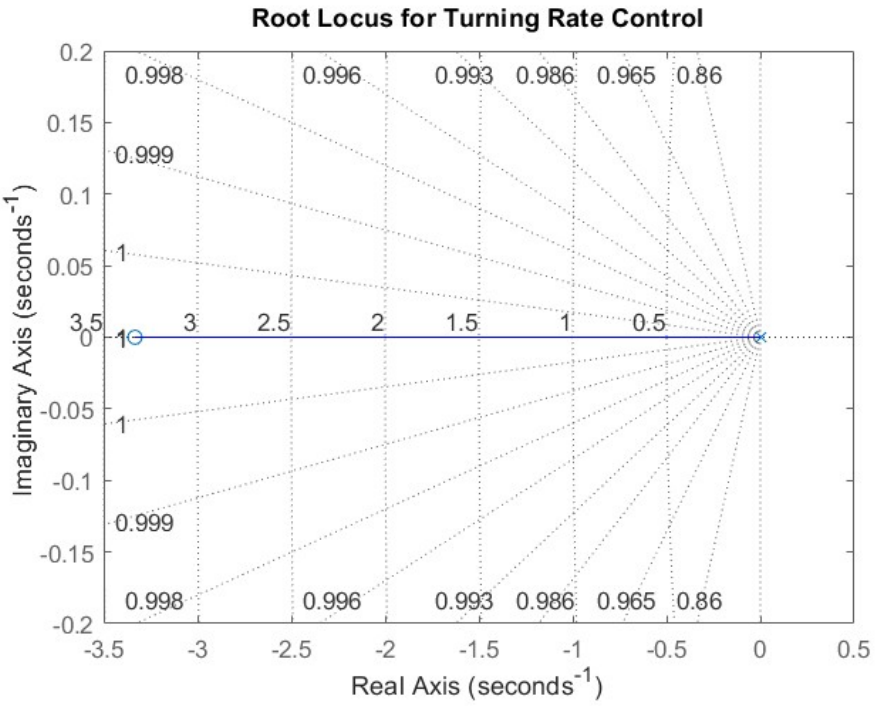

Figure 3.6: Root Locus for Turning Rate Control: The plot depicts the path of closed-loop poles for turning rate control with increasing feedback gain K. It shows the poles starting from the origin and moving towards the zero defined by $\frac{K_i}{K_p}$.

In the Root Locus Fig. 3.5 and Fig. 3.6, the horizontal axis represents the real part of the poles. Poles on the left half of the real axis indicate a stable system, while poles on the right half indicate instability. The vertical axis represents the imaginary part of the poles. Complex poles (with both real and imaginary parts) indicate oscillatory behavior in the system. The "x" marks on the plot represent the location of the poles while "o" marks the location of the zeros. As we increase the gain of the system K, these plots trace the movement of the poles from the origin towards their final positions near the zeros defined by $\frac{K_i}{K_p}$.

Based on the Root Locus analysis, we can select K_p and K_i values that provide the best performance. Poles closer to the negative real axis indicate better damping and less oscillation, and all poles should remain in the left-half-plane for stability.

For the Root Locus for velocity control, we chose the gain values as:

$$K_p = 0.5, \quad K_i = 1.5 \tag{3.27}$$

For the Root Locus for turning rate control, we chose the gain values as:

$$K_p = 0.3, \quad K_i = 1.0 \tag{3.28}$$

We can then find our closed loop poles from Eq. 3.22 and Eq. 3.23.

$$s_v = \frac{-G_v K_i}{1 + G_v K_p} = \frac{-0.2159(1.5))}{1 + 0.2159(0.5)} = -0.2923 \tag{3.29}$$

$$s_\omega = \frac{-G_w K_i}{1 + G_w K_p} = \frac{-0.238(1.0)}{1 + 0.238(0.3)} = -0.2221 \tag{3.30}$$

For velocity control, the pole at -0.2923 suggests a stable system with a moderate response speed. For turning rate control, the pole at -0.2221 also indicates stability with a slightly slower response compared to velocity control due to the smaller magnitude of the pole. Both of the negative values ensure that any perturbations in the system will exponentially decay, leading to a steady state over time without oscillations.

Our chosen gain values both satisfy the stability criteria for the closed-loop

system and adhere to the Nyquist sampling theorem. We particularly set our sampling frequency to be 6 times faster than the highest frequency of the system's response to ensure that the sampling rate is sufficient to capture the dynamics of the system accurately without inducing aliasing or missing critical behavior between sample points.

The PI controllers algorithm is updated as follows:

Algorithm 2 PI Controllers with Feedforward

Initialize global variables for PI controllers
$G_v \leftarrow 0.2159$ ▷ Plant constant for velocity
$G_w \leftarrow 0.238$ ▷ Average plant constant for turning rate
while simulation is running **do**
    Set flags $turnRateControl \leftarrow 1$, $velocityControl \leftarrow 1$
    if $turnRateControl == 1$ **then**
        $Krot_p \leftarrow 0.3$
        $Krot_i \leftarrow 1.0$
        $wFF \leftarrow wRef/G_w$ ▷ Adjusted feedforward gain
        $intRot \leftarrow intRot + (wRef - wMes) \times 50/1000$ ▷ Integrator dt = 50ms
        $uOmega \leftarrow Krot_p \times (wRef - wMes) + Krot_i \times intRot + wFF$
    else
        $uOmega \leftarrow 0$
    end if
    if $velocityControl == 1$ **then**
        $Kvel_p \leftarrow 0.5$
        $Kvel_i \leftarrow 1.5$
        $uFF \leftarrow uRef/G_v$ ▷ Adjusted feedforward gain
        $intVel \leftarrow intVel + (uRef - uMes) \times 50/1000$ ▷ Integrator dt = 50ms
        $uV \leftarrow Kvel_p \times (uRef - uMes) + Kvel_i \times intVel + uFF$
    else
        $uV \leftarrow 0$
    end if
    Send left and right motor velocities
    sim.setJointTargetVelocity($motorLeft1, -(uV - uOmega)$)
    sim.setJointTargetVelocity($motorLeft2, -(uV - uOmega)$)
    sim.setJointTargetVelocity($motorRight1, -(uV + uOmega)$)
    sim.setJointTargetVelocity($motorRight2, -(uV + uOmega)$)
end while

Another thing worth noticing is that the graph 3.4 for the turning rate step

response shows a slightly different behavior than that of the velocity step response. The control of angular velocity appears more challenging as evidenced by the oscillation in the output signal. The fluctuations seen in the graph are typical signs of a system experiencing issues such as under-damping or parameter mismatches which might be caused by the skidding of the tires.

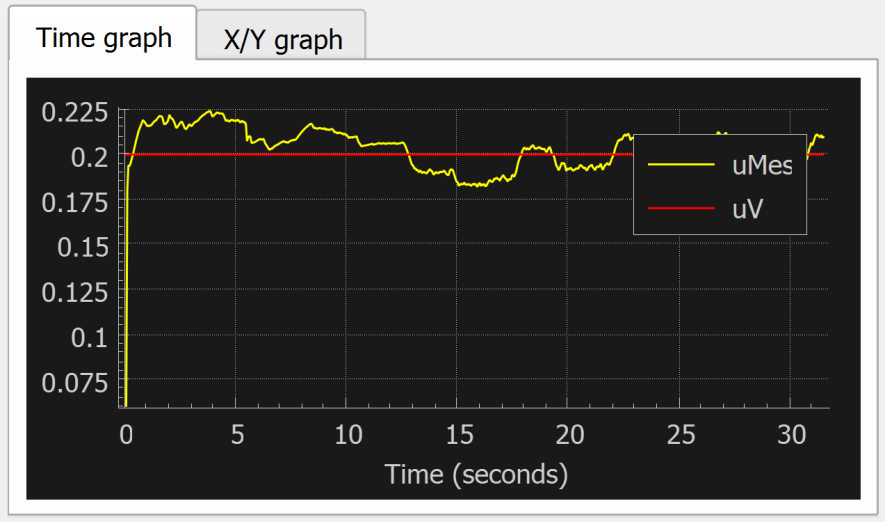

Figure 3.7: Step Response of the Velocity PI Controller given Constant Velocity and Turning Rate ($uRef = 0.2$, $wRef = 0.15$) (Tractor Mass: 29kg)

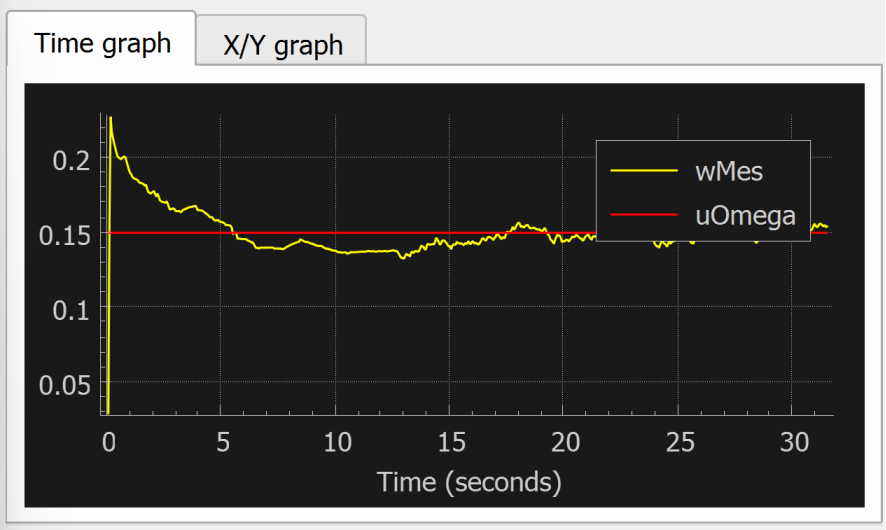

Figure 3.8: Step Response of the Turning Rate PI Controller given Constant Velocity and Turning Rate ($uRef = 0.2$, $wRef = 0.15$) (Tractor Mass: 29kg)

The skidding phenomenon during an experimental turning maneuver is notable in Fig. 3.7 and Fig. 3.8. Our tractor has a differential drive system which naturally exhibits skidding during turns because the wheels on the inside and outside of the turn need to rotate at different speeds to execute a turn. The interaction between the tires and the driving surface also significantly affects skidding. One effective approach to improve traction and reduce skidding is to increase the weight of the tractor.

The most direct effect of increasing the tractor's weight is the increase in normal force exerted by the tractor on the ground due to gravity. The normal force impacts the frictional force available between the tires and the driving surface. According to the basic principle of friction:

$$F_{friction} = \mu \times F_{normal} \tag{3.31}$$

where μ is the coefficient of friction. An increase in normal force results in a proportional increase in the maximum frictional force that can be generated.

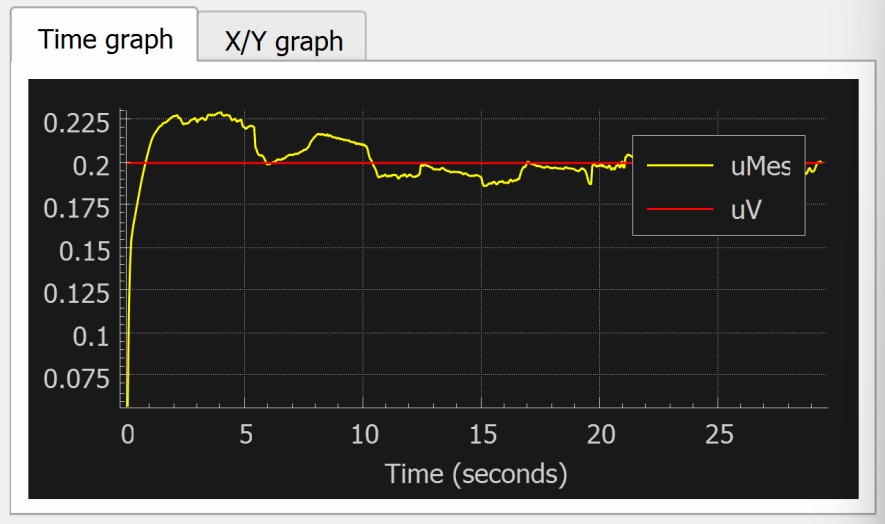

Figure 3.9: Step Response of the Velocity PI Controller given Constant Velocity and Turning Rate ($uRef = 0.2$, $wRef = 0.15$) (Tractor Mass: 129kg)

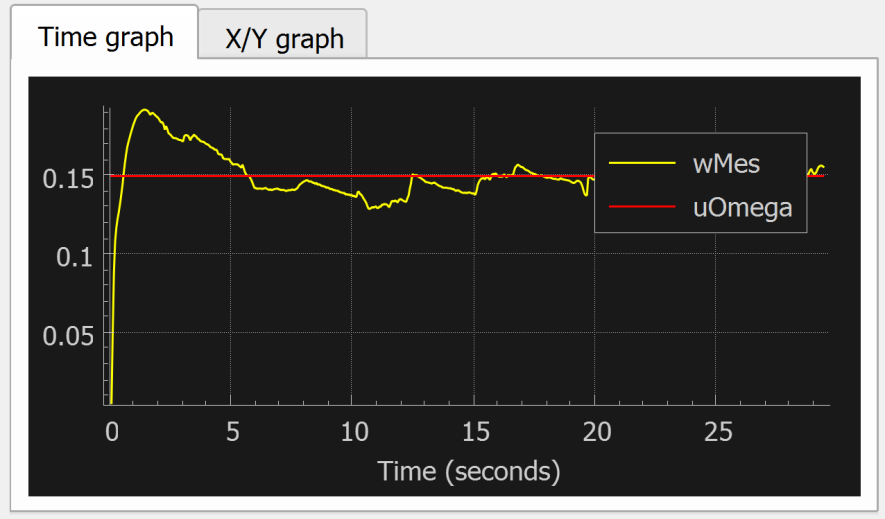

Figure 3.10: Step Response of the Turning Rate PI Controller given Constant Velocity and Turning Rate ($uRef = 0.2$, $wRef = 0.15$) (Tractor Mass: 129kg)

With greater weight, the tires of the tractor can achieve better traction as

shown in Fig. 3.9 and Fig. 3.10 after we increased the tractor's mass by 100kg.

In this chapter, we applied root locus analysis to determine the proportional (K_p) and integral (K_i) gains for the velocity and turning rate PI controllers. We verified the stability conditions and ensured that the poles of the closed-loop system lie in the left half of the s-plane. The gains were then implemented in the CoppeliaSim environment, considering the appropriate feedforward terms based on the experimentally determined plant constants. This analysis ensures that the tractor navigation system performs optimally and remains stable during operation. Additionally, we ensured that the sampling rate is adequate for the system's dynamics, maintaining a sampling rate much higher than the system's bandwidth for accurate control.

Chapter 4

Nonlinear Controller for Closed Loop Steering

In order to ensure that the tractor not only moves towards the target gate location but does so in a stable and convergent manner, our approach is to use the principles from Lyapunov function to steer the tractor. We start with a MATLAB simulation based on the unicycle model [1] to provide a practical and interactive visualization of the tractor's navigation capabilities. Then we transfer the simulation in our CoppeliaSim environment.

The MATLAB model 4.3 utilizes a unicycle kinematic model to describe the

motion of the tractor. The model is

$$\begin{aligned} \dot{x} &= u\cos\phi \\ \dot{y} &= u\sin\phi \\ \dot{\phi} &= \omega \end{aligned} \tag{4.1}$$

where (x, y) represents the position coordinates of the unicycle with respect to the target frame <g>, and ϕ denotes its orientation. The triple (x, y, ϕ) is also called the pose [2].

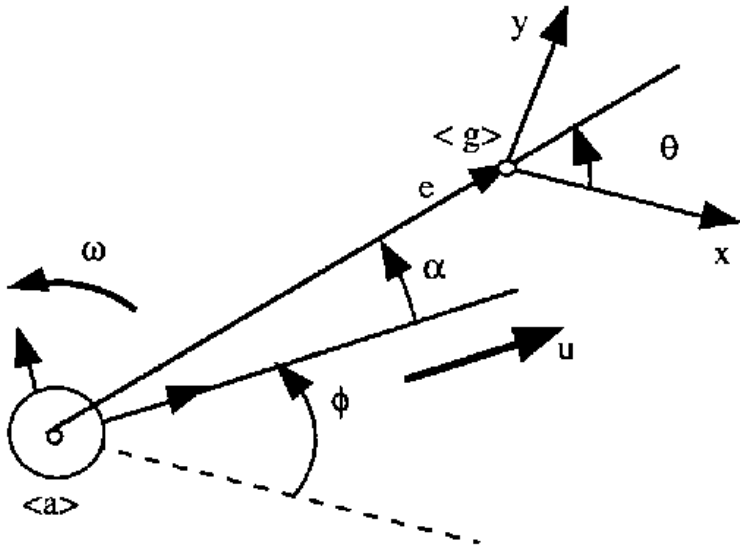

Figure 4.1: Unicycle(Tractor) Model's Position and Orientation with respect to the Target Frame [1]

The variables e, α, and θ are depicted in Fig. 4.1, and they describe the relative position between the unicycle (tractor) and a desired final (or target) pose. The model can be further derived to represent the position of the vehicle in terms of its polar

coordinates as well as error distance $e > 0$ and orientation θ with respect to the target frame <g>:

$$\begin{aligned} \dot{e} &= -u\cos(\theta - \phi) \\ \dot{\theta} &= u\frac{\sin\alpha}{e} \\ \dot{\phi} &= \omega \end{aligned} \tag{4.2}$$

Since $\alpha = \theta - \phi$ is the angle between the unicycle's principal axis (the direction of linear vector u) and the distance vector e, we have

$$\begin{aligned} \dot{e} &= -u\cos\alpha \\ \dot{\theta} &= u\frac{\sin\alpha}{e} \\ \dot{\alpha} &= u\frac{\sin\alpha}{e} - \omega \end{aligned} \tag{4.3}$$

In this model, we define the state variables of the unicycle to be $[e, \alpha, \theta]'$. The control variables are linear velocity (u) and angular velocity (ω). u and ω are dynamically adjusted based on the unicycle's instantaneous state and its relation to the target pose. The control strategy to find a state dependent control law $[u, \omega]' = g(e, \alpha, \theta)$ involves calculating the relative orientation of the unicycle with respect to the target and the angular displacement between the unicycle's current heading and the direction towards the target.

In order to guarantee the state $[e, \alpha, \theta]'$ to be asymptotically driven to the

null limiting point $[0,0,0]'$ in a stable trajectory without achieving $e=0$ in finite time, we apply the Lyapunov stability theory. Consider the following candidate Lyapunov function:

$$V_1 = \frac{1}{2}\lambda e^2 \tag{4.4}$$

$$V_2 = \frac{1}{2}(\alpha^2 + h\theta^2) \tag{4.5}$$

$$V = V_1 + V_2 = \frac{1}{2}\lambda e^2 + \frac{1}{2}(\alpha^2 + h\theta^2); \quad \lambda, h > 0 \tag{4.6}$$

where $\dot{V_1}$, $\dot{V_2}$ represent $\frac{1}{2}$ of the squared weighted norms of both the error distance vector e and the alignment error vector $[\alpha, \sqrt{h}\theta]'$ with respect to the target frame <g>.

Now, let's consider the time derivative $\dot{V}$ along 4.3

$$\dot{V_1} = \lambda e\dot{e} = -\lambda e u \cos\alpha \tag{4.7}$$

$$\dot{V_2} = a\dot{a} + h\theta\dot{\theta} = \alpha\left(-\omega + u\frac{\sin\alpha}{\alpha}\cdot\frac{\alpha + h\theta}{e}\right) \tag{4.8}$$

$$\dot{V} = \dot{V_1} + \dot{V_2} = -\lambda e u\cos\alpha + \alpha\left(-\omega + u\frac{\sin\alpha}{\alpha}\cdot\frac{\alpha + h\theta}{e}\right) \tag{4.9}$$

Eq. 4.7 can be made non-positive by letting the linear velocity u have the smooth form that is independent from both λ and h.

$$u = (\gamma\cos\alpha)e; \quad \gamma > 0 \tag{4.10}$$

Therefore, we have

$$\dot{V}_1 = -(\lambda \cos^2 \alpha)e^2 \leq 0 \tag{4.11}$$

$$\dot{V}_2 = \alpha \left(-\omega + \gamma \frac{\cos \alpha \sin \alpha}{\alpha} (\alpha + h\theta) \right) \tag{4.12}$$

This shows that $\dot{V}_1$ is always non-increasing in time and thus asymptotically converging toward a non-negative finite limit since it's lower bounded by zero.

Eq. 4.12 can also be made non-positive by letting the angular velocity ω take on the smooth form that is independent from λ but not from h.

$$\omega = k\alpha + \gamma \frac{\cos \alpha \sin \alpha}{\alpha} (\alpha + h\theta); \quad k > 0 \tag{4.13}$$

Thus, Eq. 4.12 becomes

$$\dot{V}_2 = -k\alpha^2 \leq 0 \tag{4.14}$$

The global Lyapunov function's time derivative thereby becomes a negative semi-definite form:

$$\dot{V} = \dot{V}_1 + \dot{V}_2 = -\lambda(\gamma \cos^2 \alpha)e^2 - k\alpha^2 \leq 0 \tag{4.15}$$

Eq. 4.15 confirms that the Lyapunov function V is always non-increasing in time and thus asymptotically converging toward a finite non-negative limit because of its lower bound by 0.

We therefore arrive at the control law that will be used to compute the

instantaneous turning rate and velocity of the unicycle(tractor): [1]

$$\begin{aligned} \omega &= k\alpha + \gamma \frac{\cos\alpha \sin\alpha}{\alpha}(\alpha + h\theta); \quad k > 0 \\ u &= (\gamma \cos\alpha)e; \quad \gamma > 0 \end{aligned} \tag{4.16}$$

The variables $\gamma > 0$, $k > 0$ and $h > 0$ are constant parameters of the nonlinear control law. Note that for $\alpha = 0$ the expression for ω above is undefined so we need to rewrite it as

$$\omega = k\alpha + \gamma \cos\alpha(\alpha + h\theta); \quad k > 0,\ \alpha \approx 0 \tag{4.17}$$

in which we use $\lim_{\alpha\to 0} \frac{\sin\alpha}{\alpha} = 1$.

The control parameters k, γ, and h in the nonlinear controller 4.16 are essential to determine the behavior and effectiveness of the control system used to steer a unicycle-like vehicle.

The parameter k is the proportional gain for turning rate ω, and it affects the responsiveness of the turning rate to the angular error α, which is the angle between the unicycle's current heading and the target direction (see Fig. 4.1). A higher k value results in a more aggressive turning response, which can help the vehicle align with the target direction more quickly but might lead to overshooting or instability if too high. On the other hand, a lower k value provides a gentler response, which can be too slow if the vehicle needs to rapidly change direction and lead to sluggish behavior.

The parameter γ is a gain in both turning rate and velocity equations that scales the influence of the cosine component in both equations, adjusting how alignment

with the target direction affects the control outputs. A larger γ enhances the sensitivity of both velocity and turning rate to the alignment with the target direction. It can lead to higher speeds and sharper turns when the vehicle is well-aligned with the target. A smaller γ reduces these sensitivities and can prevent abrupt changes in speed and direction but may also slow down the correction of the vehicle's path towards the target.

Finally, the parameter h is an additional augmentation factor that modifies how the heading error θ (the angle to the target pose from the unicycle's current position) influences the turning rate, alongside α. A higher h value increases the contribution of the heading error θ to the turning control, potentially useful in situations where the vehicle needs to consider its relative position to the target more significantly. A lower h value minimizes the influence of θ and focuses the control more directly on the vehicle's orientation α with respect to the target.

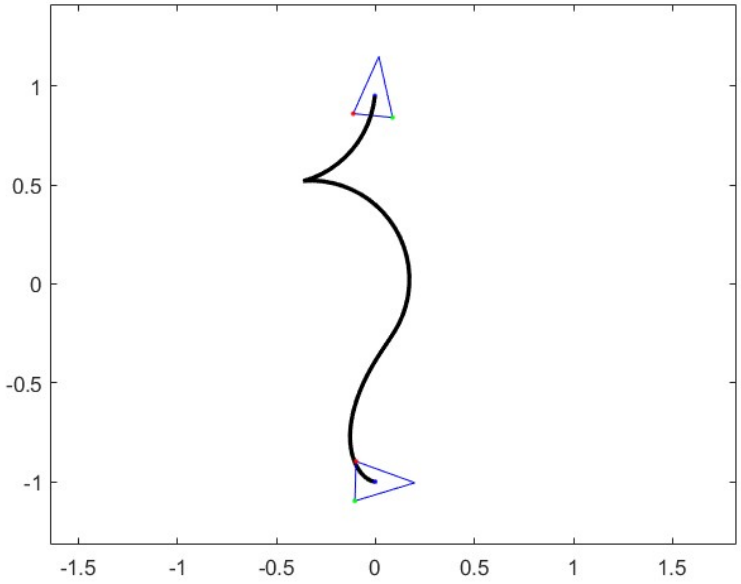

Figure 4.2: Unicycle Navigation Example (Starting Pose: $[0, 1, 90°]$; Target Pose: $[0, -1, 0°]$; $k = 6$, $\gamma = 3$, $h = 1$)

The selection of these parameters involves a combination of empirical turning and theoretical derivation based on the dynamics of the vehicle and the desired control characteristics. Parameters can be initially set based on the vehicle's size, weight, maximum speed, and turning capabilities. Adjustments can be made based on the observed performance in simulation.

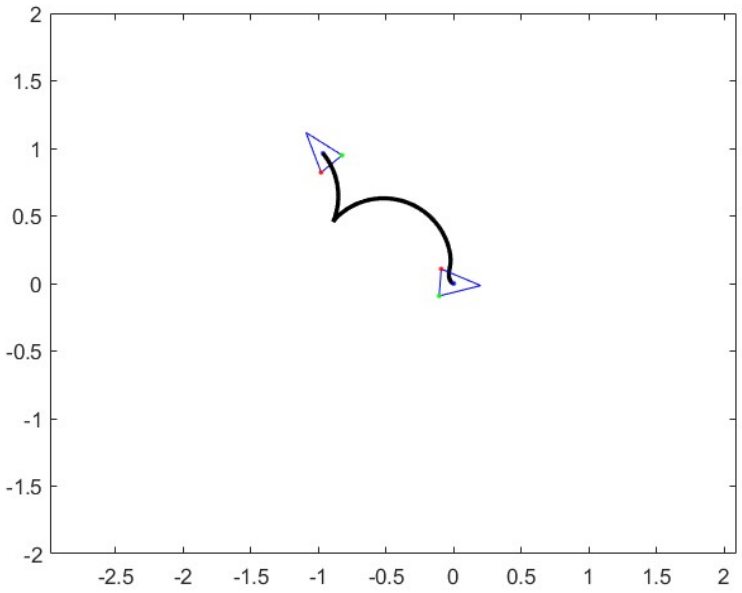

Figure 4.3: Unicycle Navigation Example (Starting Pose: $[-1, 1, 135°]$; Target Pose: $[0, 0, 0°]$)

The CoppeliaSim VREP simulated environment has been set up to implement tractor navigation simulation. The scene features a tractor and two cylindrical poles which represent the gate posts. The objective of this simulation is to allow the user to visualize the tractor navigating towards the center of the gate posts on demand.

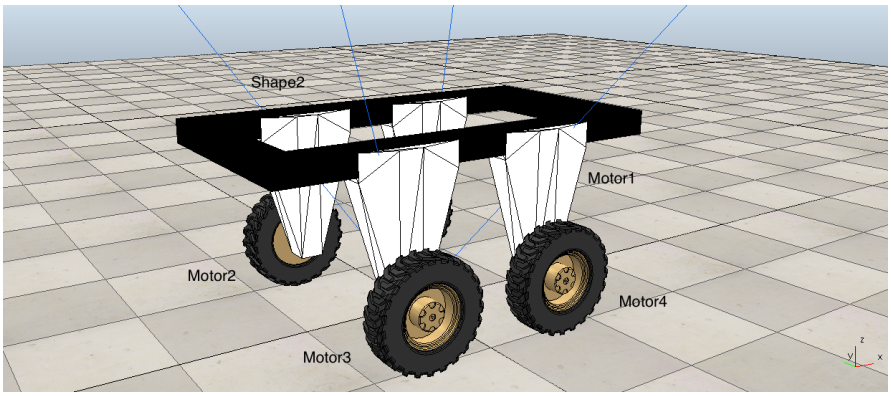

Figure 4.4: Scene of the Tractor in CoppeliaSim

The tractor's body, labeled `Shape2` in the scene hierarchy, is equipped with four motorized wheels: `Motor1`, `Motor2`, `Motor3`, and `Motor4`. `Motor2` and `Motor3` are located in the front of the tractor, while `Motor1` and `Motor4` driving the rear wheels of the tractor. Therefore, the right wheels of the tractor are controlled by `Motor3` and `Motor4`, and the left wheels are driven by `Motor2` and `Motor1`. These modeled motors make up the differential driving dynamics of the tractor and are controlled to achieve the desired trajectory and orientation.

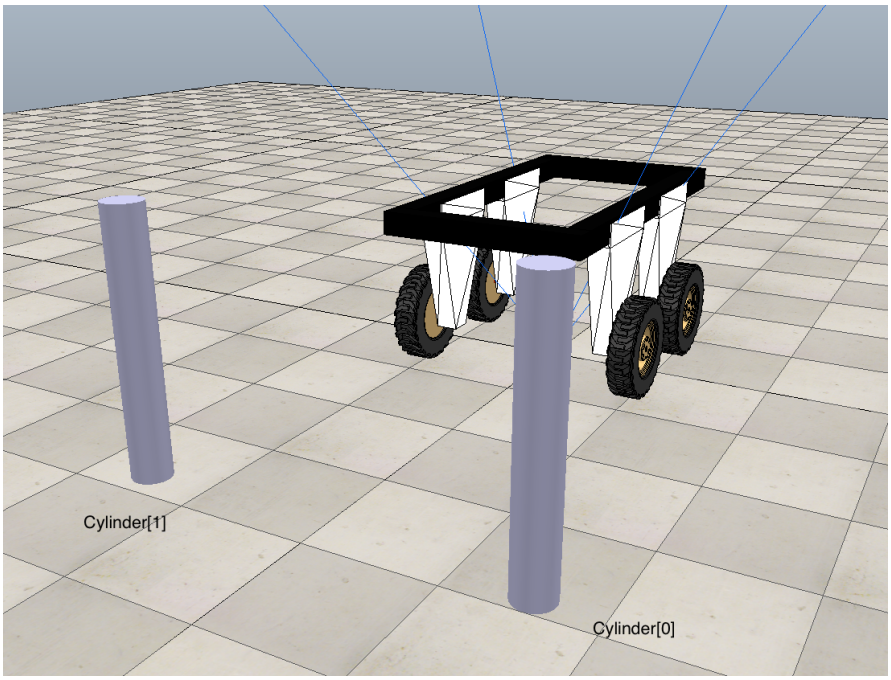

Figure 4.5: The Tractor and the Gate Posts in CoppeliaSim

The two cylindrical poles referred to as `Cylinder[0]` and `Cylinder[1]` in the scene hierarchy as shown in Fig. 4.5, are static objects placed at a specific distance from each other to create a gate through which the tractor is intended to park into. The position of these two cylinders in the world frame is used to calculate the target pose, including the location and orientation that the tractor must achieve to successfully park.

The feedback control strategy takes the tractor's current state consisting of position and orientation to compute the necessary control inputs (velocity and turning rate) that will steer the tractor toward the parking pose which is in the middle of the parking gate and have the heading orthogonal to the line connecting the posts.

We first calculate the relative orientation of the tractor with respect to the

target pose and the angle from the tractor's current position to the target, as they are needed for determining how the tractor should turn to face the target. The control inputs for velocity and turning rate are then calculated and adjusted using the control law 4.16 based on the tractor's proximity to the target and the alignment with the target orientation.

In our VREP simulation, the `sysCall_init` function contains the foundational setup, including retrieving the tractor's motor handles for actuation and defining the initial and target poses. The orientation of the target gate is also calculated by identifying the midpoint between the two gate posts and establishing a perpendicular direction vector to ascertain the gate's facing direction.

The `sysCall_actuation` function is structured to compute the control inputs necessary for navigation based on the tractor's current and desired states. This function directly influences the tractor's behavior by dynamically adjusting the motor speeds to match the desired trajectory and orientation as determined by the navigation strategy. The nonlinear controller is implemented here to determine the appropriate velocity (`uRef`) and turning rate (`wRef`) that are subsequently managed by the PI controller to command the tractor's motor speeds.

The nonlinear controller begins by checking the magnitude of the angular difference (`alpha`) between the tractor's current orientation and the desired orientation toward the target. Depending on the value of `alpha`, the function decides between two formulas to compute `wRef`. If `alpha` is significant, we use the control law 4.16 to account for both the angular error and its rate of change to provide a dynamic adjustment to

the tractor's orientation.

$$\texttt{wRef} = k\alpha + \gamma \cos(\alpha) \cdot \frac{\sin(\alpha)}{\alpha}(\alpha + h\theta) \tag{4.18}$$

If `alpha` is nearly zero (less than a very small threshold, here `1e-8`), implying that the tractor is almost facing the target, We'll use 4.17:

$$\texttt{wRef} = k\alpha + \gamma \cos(\alpha) \cdot (\alpha + h\theta); \quad (a \approx 0) \tag{4.19}$$

We then subsequently calculate the velocity based on both the alignment and the remaining distance to the target, slowing down as the tractor aligns with the target and approaches it 4.16.

$$\texttt{uRef} = (\gamma \cos(\alpha))e \tag{4.20}$$

For our VREP tractor, we set $k = \frac{3.0}{50}$ to provide a balanced approach, offering sufficient responsiveness without causing instability in the vehicle's turning behavior. We chose $\gamma = \frac{3.0}{10}$ to ensure smooth deceleration as the tractor approaches the target pose and maximize the accuracy of stopping at the desired location. We set $h = \frac{3.0}{1}$ to allow for subtle corrections when navigating towards the target with high precision. These values are empirically determined through simulation trials to achieve a balance between responsiveness and stability in the tractor's navigation system.

Algorithm 3 Nonlinear Controller Logic

if abs(alpha) $< 1e-8$ **then**
    wRef $\leftarrow k \cdot \text{alpha} + \gamma \cdot \cos(\text{alpha}) \cdot (\text{alpha} + h \cdot \text{theta})$
    uRef $\leftarrow \gamma \cdot \cos(\text{alpha}) \cdot \text{distance}$
else
    wRef $\leftarrow k \cdot \text{alpha} + \gamma \cdot \cos(\text{alpha}) \cdot \frac{\sin(\text{alpha})}{\text{alpha}} \cdot (\text{alpha} + h \cdot \text{theta})$
    uRef $\leftarrow \gamma \cdot \cos(\text{alpha}) \cdot \text{distance}$
end if

In `sysCall_actuation` function, we program the nonlinear controller by defining `wRef` and `uRef`, as shown in Algorithm 3.

Conditional overrides for both turning rate and velocity are implemented to allow for explicit control under certain conditions.

Algorithm 4 Conditional Control Overrides

if `turnRateControl` $= 0$ **then**
    $wRef \leftarrow 0$ ▷ Disable turning if turn rate control is disabled
end if
if `velocityControl` $= 0$ **and** `turnRateControl` $= 1$ **then**
    $angdiff \leftarrow \text{atan2}(\sin(angdiff), \cos(angdiff))$ ▷ Normalize angular difference
    if $angdiff > 0$ **then**
        $wRef \leftarrow -0.1$ ▷ Adjust to turn left
    else if $angdiff < 0$ **then**
        $wRef \leftarrow 0.1$ ▷ Adjust to turn right
    else
        $wRef \leftarrow 0$ ▷ No turning adjustment needed
    end if
    $uRef \leftarrow 0$ ▷ Stop forward movement
end if

When the `turnRateControl` flag is set to 0, the turning rate reference `wRef` is forced to zero to halt any rotational movement of the tractor regardless of other calculations or sensor readings. When `velocityControl` flag is set to 0, the tractor's forward velocity reference `uRef` is set to zero to prevent it from moving forward. Meanwhile,

if `turnRateControl` is still active (set to 1), the tractor can adjust its orientation to properly align with a desired heading or target direction. The adjustment of `wRef` based on the angular difference $angdiff$ ensures that the tractor can minutely correct its orientation by rotating slightly left or right with a controlled rate (-0.1 or 0.1), or stop rotating if it is already well-aligned ($angdiff$ near zero).

Following the execution of the conditional overrides based on the system's current state and control flags, control is handed over to our Proportional-Integral (PI) controllers.

The `sysCall_sensing` function is called once every simulation step after the physics calculations are finished. It contains the code to read sensor data and the state of objects after they have responded to control inputs from the previous simulation step. State variables are updated in this function for the next control decisions.

Firstly, the tractor's current position and orientation in the world frame are acquired in this function. Function `sim.getObjectPosition(tractorBody, -1)` retrieves the global position coordinates (x, y, z) of the tractor. Function `sim.getObject Orientation(tractorBody, sim.handle_world)` fetches the tractor's orientation in Euler angles relative to the global reference frame. These angles are then converted into yaw, pitch, and roll using the `sim.alphaBetaGammaToYawPitchRoll` function, providing a practical angle (yaw) for directional computations.

Next, the position and heading extracted are then used to update the `currentPose` structure, which holds the tractor's real-time x, y coordinates and its heading (yaw), thus keeping the navigation system informed of the tractor's exact

whereabouts and orientation in the simulation. The Euclidean distance from the tractor to the target position is computed, giving a scalar value that indicates how far the tractor is from its desired endpoint. The orientation and angles for the next control decision are updated by reading Φ and Θ, where Φ is the relative orientation of the tractor with respect to the target pose, and Θ is the angle from the tractor's current position to the target pose. Both Φ and Θ are normalized to be within the range $[-\pi, \pi]$ to avoid issues that occur when angles wrap around. α is the angle between the tractor's current orientation and the direction towards the target pose, and it's also updated in this function.

The function also acquires the tractor's current linear and angular velocities using `sim.getVelocity(tractorBody)` function. Linear velocity `uMes` is calculated by projecting the velocity components onto the tractor's heading direction. Angular velocity `wMes` is extracted directly as the angular velocity around the vertical axis, representing the tractor's turning rate. If the tractor is sufficiently aligned with the target ($angdiff$ small enough), both velocity and turning rate references (`uRef`, `wRef`) are set to zero to halt the tractor precisely at the desired location and orientation.

Algorithm 5 Sensing Function

Update current position and heading:
tractorPos ← sim.getObjectPosition(tractorBody, -1)
orientation ← sim.getObjectOrientation(tractorBody, sim.handle_world)
atmp, btmp, ctmp ← orientation[1], orientation[2], orientation[3]
yaw, pitch, roll ← sim.alphaBetaGammaToYawPitchRoll(atmp, btmp, ctmp)
currentPose ← {tractorPos[1], tractorPos[2], yaw}
For nonlinear controller:
Compute distance and angles:
dx ← targetPose.x − currentPose.x
dy ← targetPose.y − currentPose.y
distance ← sqrt($dx^2 + dy^2$)
phi ← currentPose.heading − targetPose.phi
theta ← atan2(dy, dx) − targetPose.phi
theta ← atan2(sin(theta), cos(theta))
alpha ← theta − phi
alpha ← atan2(sin(alpha), cos(alpha))
Read turning rate and velocity:
velMes, rateMes ← sim.getVelocity(tractorBody)
uMes ← velMes[1] · cos(currentPose.heading) + velMes[2] · sin(currentPose.heading)
wMes ← rateMes[3] ▷ Axis for yaw rate
Manage close proximity and final alignment:
if distance ≤ 0.1 **and** turnRateControl = 1 **then**
    uRef ← 0
    velocityControl ← 0
    angdiff ← currentPose.heading − targetPose.phi
    angdiff ← atan2(sin(angdiff), cos(angdiff))
    if angdiff > 0 **then**
        wRef ← −0.1
    else if angdiff < 0 **then**
        wRef ← 0.1
    else
        wRef ← 0
    end if
    if abs(angdiff) ≤ 0.0349 **then** ▷ Approximately 2 degrees
        wRef ← 0
        turnRateControl ← 0
    end if
end if
Debugging and data monitoring:
Print current pose and angle differences
Update graph streams with current and reference values

Fig. 4.6 demonstrates an example of a successful application of the nonlinear control strategy to navigate the tractor toward a designated gate. The trajectory, indicated by the red line, shows the path followed by the vehicle from its starting point to the target gate location. Notably, the trajectory is smooth and direct, which illustrates the controller's efficiency in minimizing path deviations and achieving a precise alignment with the target. The precise convergence of the tractor to the target, as evidenced in the simulation, validates the theoretical aspects covered in this chapter.

Navigation example 1:

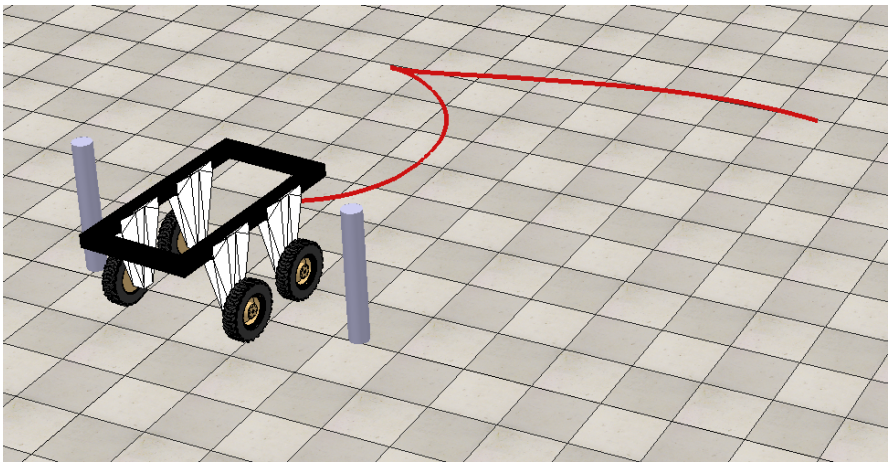

Figure 4.6: Trajectory of the Tractor Navigating from Starting Pose $[0, 0, 180°]$ to Target Pose $[5, 5, 90°]$ in CoppeliaSim

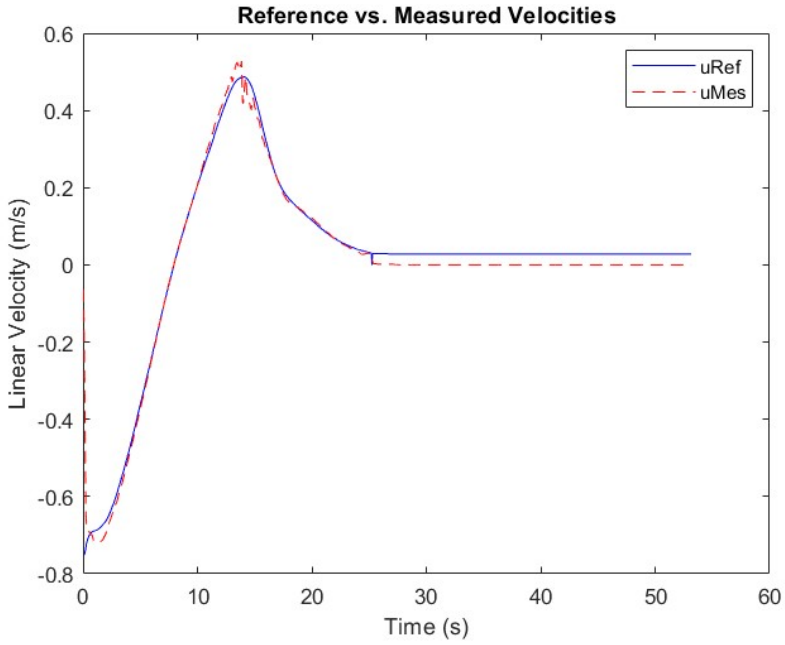

Figure 4.7: Referenced vs. Measured Velocities of the Trajectory in Fig. 4.6

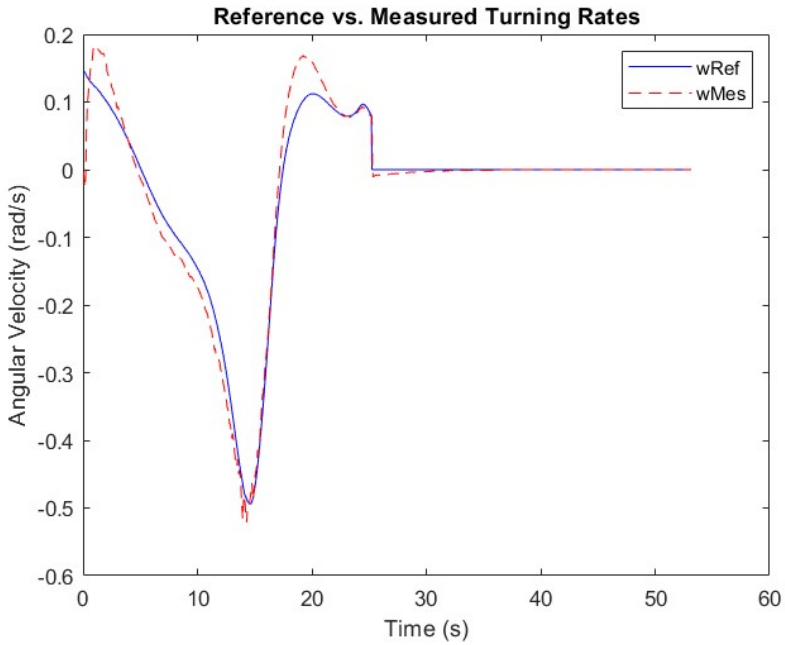

Figure 4.8: Referenced vs. Measured Turning Rates of the Trajectory in Fig. 4.6

Fig. 4.7 and Fig. 4.8 illustrate the response of the velocity and turning rate PI controllers given the example trajectory shown in Fig. 4.6. This further verifies that our PI controllers have been designed successfully to help the tractor execute velocity and turning rate controls to follow the nonlinear controller's trajectory.

Navigation example 2:

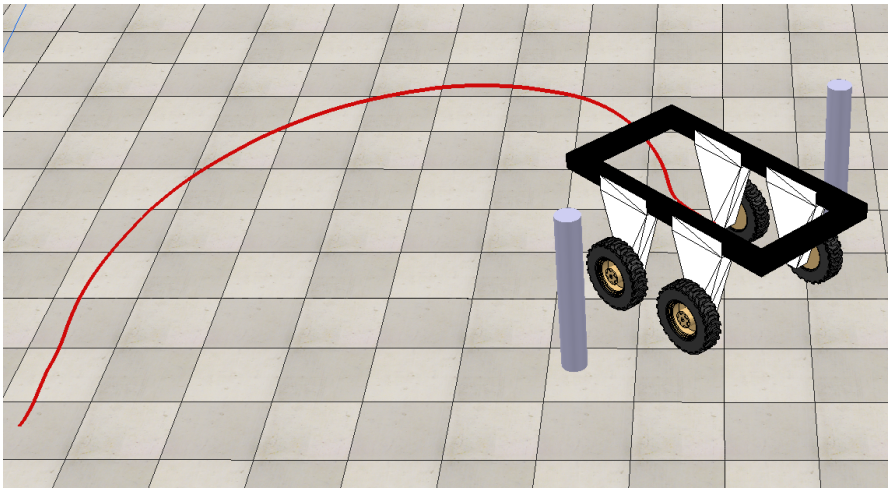

Figure 4.9: Trajectory of the Tractor Navigating from Starting Pose $[0, 0, 180°]$ to Target Pose $[0, 5, 45°]$ in CoppeliaSim

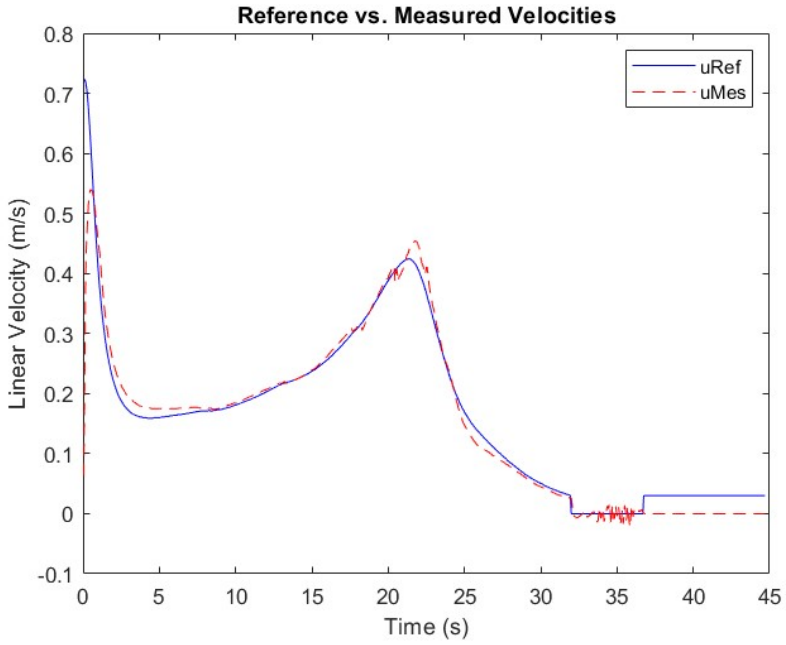

Figure 4.10: Referenced vs. Measured Velocities of the Trajectory in Fig. 4.9

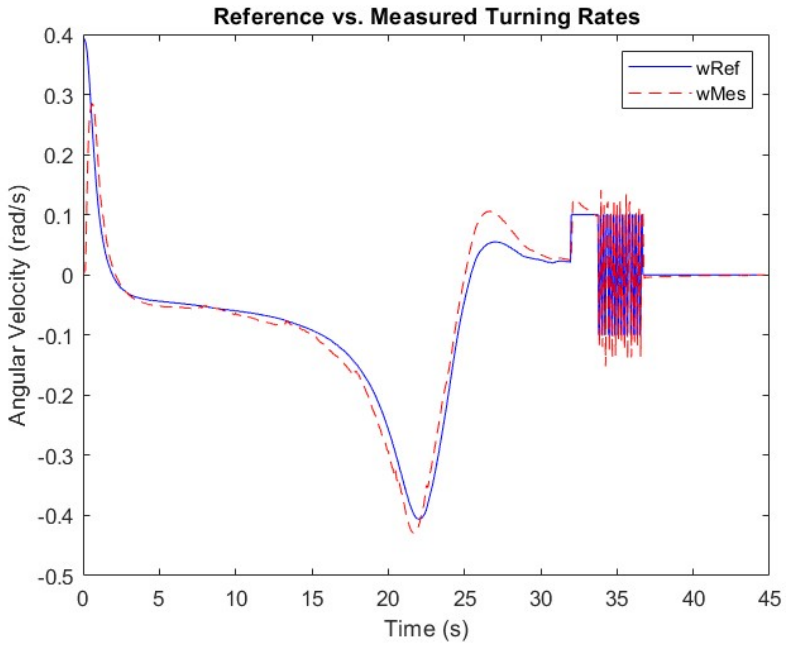

Figure 4.11: Referenced vs. Measured Turning Rates of the Trajectory in Fig. 4.9

Navigation example 3:

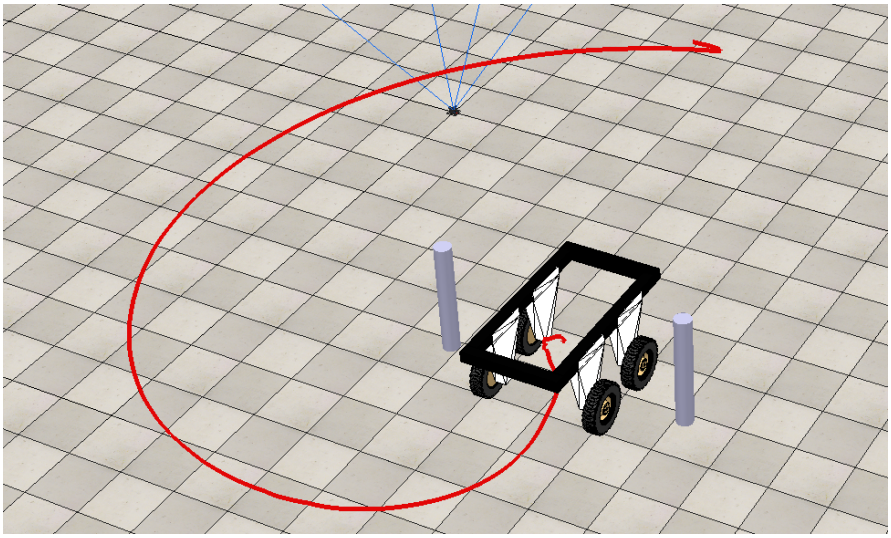

Figure 4.12: Trajectory of the Tractor Navigating from Starting Pose $[0, 0, 180°]$ to Target Pose $[3, -3, 90°]$ in CoppeliaSim

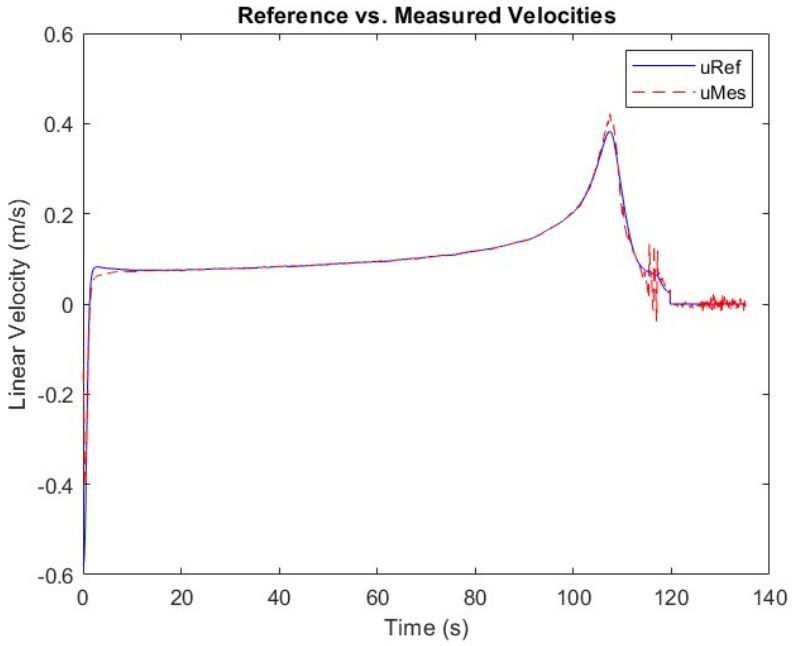

Figure 4.13: Referenced vs. Measured Velocities of the Trajectory in Fig. 4.12

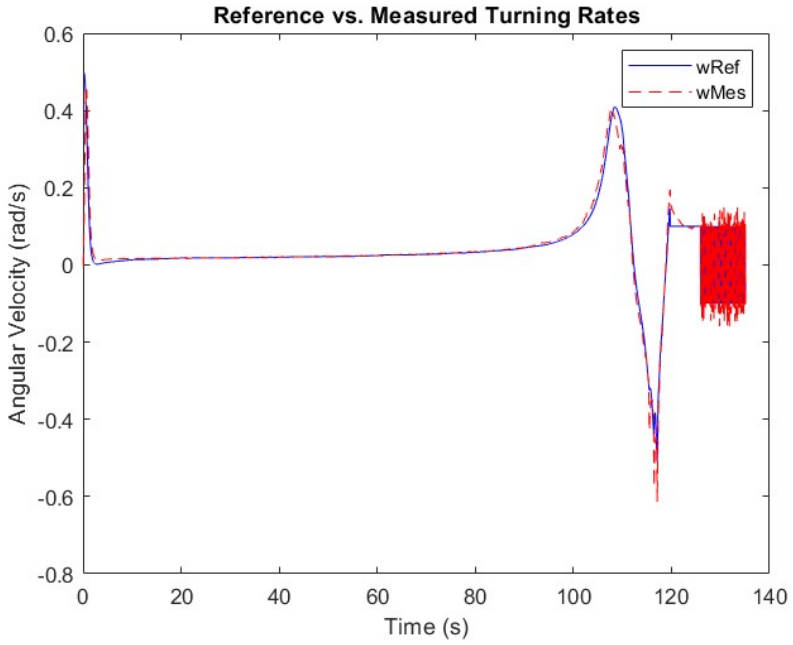

Figure 4.14: Referenced vs. Measured Turning Rates of the Trajectory in Fig. 4.12

In this chapter, we used the control law 4.16 and the MATLAB unicycle model to help us establish our nonlinear controller and provide reference steering inputs `uRef` and `wRef` for the tractor, which will be passed to our PI controller system to control the motors accordingly. We've also tuned parameters k, γ, and h to ensure that the tractor behaves in a predictable manner, maintaining a smooth trajectory while effectively adjusting its path and speed to reach and align with the target gate accurately.

Chapter 5

Conclusion

This thesis has explored the integration and application of nonlinear control techniques and Proportional-Integral (PI) controllers within a simulated environment to advance the field of autonomous vehicle navigation. The primary objective was to develop an effective navigation strategy that enables our farming tractor to perform an autonomous precise parking maneuver through the dynamic adjustment of its trajectory and orientation.

The application of Lyapunov stability theory, discussed in Chapter 2, provided a solid theoretical basis to make sure that our control strategies lead to stable and convergent navigation paths. This theory is significant in developing a control system that guarantees the stability of the autonomous tractor under all operating conditions. Chapter 4 detailed the nonlinear controller implementation and tuning to dynamically calculate the necessary control inputs. This controller effectively adjusts the tractor's trajectory towards the target, taking into account its current state relative to the desired

final position and orientation. Chapter 3 highlighted the implementation of PI controllers that translated the desired velocities and turning rates from the nonlinear controller into actual motor commands. The tuning of these controllers supported by Root Locus analysis, was critical in minimizing overshoot and ensuring the stability of the control system. The chosen gain values for the PI controllers were validated to maintain system stability and adhere to the digital Nyquist sampling theorem, providing a smooth and responsive control mechanism.

Using simulation methods such as our CoppeliaSim VREP has proven to be exceptionally beneficial in development and rigorous testing. We were able to efficiently test various control strategies, configurations, and scenarios without the need for extensive physical trials, which are often time-consuming, costly, and subject to a multitude of unpredictable variables. Adjustments to control parameters can be made in real-time and tested under a variety of conditions to observe outcomes immediately in simulation. This is significantly more cost-effective than conducting numerous real-world experiments, which require physical setup, maintenance, and often involve high costs related to wear and tear of the hardware components. The simulated environment provides a controlled setting where every variable can be meticulously managed. This is crucial when testing the impact of specific factors on the vehicle's behavior. It ensures repeatability in experiments which is a fundamental aspect often challenging to achieve in outdoor settings due to varying weather conditions, battery performance inconsistencies, and potential electronic malfunctions in the actual hardware. Engineering using simulation also reduces risks associated with testing in a real-world environment. Early-stage bugs

and errors in the control algorithms can lead to unpredictable behaviors and potential accidents when tested in the real world. We can mitigate these risks by first validating these systems in a virtual environment before real-world deployment. Additionally, the scalable nature of simulations makes them ideal for testing a wide range of scenarios that would be impractical or impossible to recreate physically. These include extreme weather conditions, different types of terrains, or emergency situations. Moreover, the simulation environment can be extended to include more complex scenarios involving multiple vehicles, obstacles, and pedestrians, providing a comprehensive platform for future research and development.

Bibliography

[1] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino. Closed loop steering of unicycle like vehicles via lyapunov techniques. *IEEE Robotics & Automation Magazine*, 2(1):27–35, 1995.

[2] Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second edition, 2017. ISBN 978-3-319-54413-7.

[3] Andreas Folkers, Matthias Rick, and Christof Büskens. Controlling an autonomous vehicle with deep reinforcement learning. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2025–2031, 2019.

[4] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2022.

[5] Jaepoong Lee and Seongjin Yim. Comparative study of path tracking controllers on low friction roads for autonomous vehicles. *Machines*, 11(3), 2023.

[6] Guo Qing, Zhang Zheng, and Xu Yue. Path-planning of automated guided vehicle based on improved dijkstra algorithm. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 7138–7143, 2017.

[7] C. Samson and K. Ait-Abderrahim. Feedback control of a nonholonomic wheeled cart in cartesian space. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1136–1141 vol.2, 1991.

[8] Moveh Samuel, Mohamed Hussein, and Maziah Binti Mohamad. A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle. *International Journal of Computer Applications*, 135(1):35–38, 2016.

[9] Steven H. Strogatz. *Nonlinear Dynamics and Chaos*. CRC Press Taylor & Francis Group, 6000 Broken Sound Parkway NW, Suite 300, Boca Raton, FL 33487-2742, 2018.

[10] Jasmin Velagic, Bakir Lacevic, and Nedim Osmic. *Nonlinear Motion Control of Mobile Robot Dynamic Model*. 06 2008.

[11] Wikipedia contributors. Bendixson's inequality — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Bendixson%27s_inequality&oldid=1196572784`, 2024. [Online; accessed 25-June-2024].

[12] Wikipedia contributors. Nyquist–shannon sampling theorem — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Nyquist%E2%80%`

93Shannon_sampling_theorem&oldid=1222782688, 2024. [Online; accessed 8-May-2024].

[13] Fitri Yakub and Yasuchika Mori. Comparative study of autonomous path-following vehicle control via model predictive control and linear quadratic control. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 229(12):1695–1714, 2015.

[14] Qin Zou, Qin Sun, Long Chen, Bu Nie, and Qingquan Li. A comparative analysis of lidar slam-based indoor navigation for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6907–6921, 2022.

Appendix A

CoppeliaSim Simulation Code for the Tractor

A.1 Initialization Script

This script sets up the simulation environment, initializes variables, and configures the simulation parameters.

```
function sysCall_init()
    -- Load necessary SIM library
    sim = require('sim')


    -- Open files for logging
    file = io.open('PI.txt', 'w+')
    file:write('Velocity and Turning Rate of the final trajectory:\n\n')
```

```
accuracy = io.open('trajectory.txt', 'w')

accuracy:write("Time, X, Y\n")


-- Initialization of global variables for poses and controls

targetPose = {x, y, phi}

currentPose = {x, y, heading}

distance = 0

turnRateControl = 1

velocityControl = 1

intRot = 0.0   -- Integrator for turning rate

intVel = 0.0   -- Integrator for velocity

uOmega = 0.0   -- Control variable for turning rate

uV = 0.0       -- Control variable for velocity

wRef = 0.0     -- Reference turning rate

uRef = 0.0     -- Reference velocity

uMes = 0.0     -- Measured velocity

wMes = 0.0     -- Measured turning rate

phi = 0        -- Relative orientation to the target

theta = 0      -- Angle to the target pose

alpha = 0      -- Angle between current and target orientation
```

```
-- Control parameters

k = 3.0/50

gamma = 3.0/10

h = 3.0/1


-- Get motor handles

motorRight1 = sim.getObjectHandle("Motor4")

motorRight2 = sim.getObjectHandle("Motor3")

motorLeft1 = sim.getObjectHandle("Motor1")

motorLeft2 = sim.getObjectHandle("Motor2")


-- Set up graph streams for debugging and visualization

handleGraph = sim.getObjectHandle('/Graph')

stream1 = sim.addGraphStream(handleGraph, 'ref_turning_rate', '',

        0, {1, 0, 0}, 0)

stream2 = sim.addGraphStream(handleGraph, 'turning_rate', '',

        0, {1, 1, 0}, 0)

handleGraph1 = sim.getObjectHandle('/Graph1')

stream3 = sim.addGraphStream(handleGraph1, 'ref_velocity', '',

        0, {1, 0, 0}, 0)

stream4 = sim.addGraphStream(handleGraph1, 'velocity', '',

        0, {1, 1, 0}, 0)
```

```
-- Add drawing object for visual trajectory tracking
trajectory = sim.addDrawingObject(sim.drawing_linestrip,
        5, 0, -1, 9000, {1, 0, 0})

-- Initial tractor state and gate calculations
tractorBody = sim.getObjectHandle("Shape2")
tractorPos = sim.getObjectPosition(tractorBody, -1)
currentOrientationEuler = sim.getObjectOrientation(tractorBody,
        sim.handle_world)
atmp = currentOrientationEuler[1]
btmp = currentOrientationEuler[2]
ctmp = currentOrientationEuler[3]
yawtmp, pitchtmp, rolltmp =
        sim.alphaBetaGammaToYawPitchRoll(atmp, btmp, ctmp)
currentPose.x = tractorPos[1]
currentPose.y = tractorPos[2]
currentPose.heading = yawtmp

-- Calculate and normalize directions and distances to the gate
positionLeft = sim.getObjectPosition(
        sim.getObjectHandle("/Cylinder[2]"), -1)
```

```
    positionRight = sim.getObjectPosition(
            sim.getObjectHandle("/Cylinder[1]"), -1)
    targetPose.x = (positionLeft[1] + positionRight[1]) / 2
    targetPose.y = (positionLeft[2] + positionRight[2]) / 2
    local gateDirection = {x = positionRight[1] - positionLeft[1],
            y = positionRight[2] - positionLeft[2]}
    local perpendicularDirection = {x = -gateDirection.y,
            y = gateDirection.x}
    targetPose.phi = math.atan2(perpendicularDirection.y,
            perpendicularDirection.x) + math.pi / 2
    distance = math.sqrt((targetPose.x - currentPose.x)^2
            + (targetPose.y - currentPose.y)^2)
    phi = currentPose.heading - targetPose.phi
    theta = math.atan2(targetPose.y - currentPose.y,
            targetPose.x - currentPose.x) - targetPose.phi
    theta = math.atan2(math.sin(theta), math.cos(theta))
    alpha = theta - phi
    alpha = math.atan2(math.sin(alpha), math.cos(alpha))
    angdiff = math.atan2(math.sin(currentPose.heading - targetPose.phi),
            math.cos(currentPose.heading - targetPose.phi))
end
```

A.2 Actuation Script

This script details the dynamic response. It evaluates the current state of the tractor and calculates the necessary control inputs (velocity and turning rate) to achieve desired motion. The script includes nonlinear and PI controller logic that dynamically adjusts motor commands based on real-time sensor data and predefined control parameters, ensuring precise and responsive maneuvering of the vehicle within the simulated environment.

```
function sysCall_actuation()
    -- Nonlinear controller determines reference
            velocities based on system state
    if (math.abs(alpha) < 1e-8) then
        wRef = k*alpha + gamma*math.cos(alpha)*(alpha + h*theta)
        uRef = gamma*math.cos(alpha)*distance
    else
        wRef = k*alpha + gamma*math.cos(alpha)*math.sin(alpha)/alpha
                *(alpha + h*theta)
        uRef = gamma*math.cos(alpha)*distance
    end

    -- Disable turning rate control if the flag is set
    if (turnRateControl == 0) then
```

```
    wRef = 0
end

-- Adjust for zero velocity control with active turning rate control
if (velocityControl == 0) and (turnRateControl == 1) then
    angdiff = math.atan2(math.sin(angdiff), math.cos(angdiff))
    if angdiff > 0 then
        wRef = -0.1
    elseif angdiff < 0 then
        wRef = 0.1
    else
        wRef = 0
    end
    uRef = 0
end

-- Plant constants for PI controllers
G_v = 0.2159  -- Plant constant for velocity
G_w = 0.238   -- Average plant constant for turning rate

-- PI controller for turning rate
if (turnRateControl == 1) or (turnRateControl == 0) then
```

```
    Krot_p = 0.3  -- Proportional gain for rotation
    Krot_i = 1.0  -- Integral gain for rotation
    intRot = intRot + (wRef - wMes)
            * 50 / 1000  -- Update integrator
    wFF = wRef / G_w  -- Feedforward term for turning rate
    uOmega = Krot_p * (wRef - wMes) + Krot_i
            * intRot + wFF  -- Output of PI controller
end

-- PI controller for velocity
if (velocityControl == 1) then
    Kvel_p = 0.5  -- Proportional gain for velocity
    Kvel_i = 1.5  -- Integral gain for velocity
    intVel = intVel + (uRef - uMes)
            * 50 / 1000  -- Update integrator
    uFF = uRef / G_v  -- Feedforward term for velocity
    uV = Kvel_p * (uRef - uMes) + Kvel_i
            * intVel + uFF  -- Output of PI controller
else
    uV = 0  -- Set velocity to zero if control is disabled
end
```

```
    -- Set motor velocities to execute motion
    sim.setJointTargetVelocity(motorLeft1, -(uV - uOmega))
    sim.setJointTargetVelocity(motorLeft2, -(uV - uOmega))
    sim.setJointTargetVelocity(motorRight1, -(uV + uOmega))
    sim.setJointTargetVelocity(motorRight2, -(uV + uOmega))
end
```

A.3 Sensing Script

This script is responsible for continuously monitoring and updating the tractor's state within the simulation environment. It retrieves and processes sensor data to accurately determine the vehicle's current position, orientation, and motion parameters. These measurements feed into the control systems to adjust the vehicle's trajectory dynamically. The script integrates non-linear control computations to ensure the vehicle's orientation and position are optimally aligned towards the desired target, facilitating precise and effective navigation responses. The detailed logging within the script aids in debugging and fine-tuning the control algorithms by providing real-time feedback on vehicle dynamics.

```
function sysCall_sensing()
    -- Retrieve current position and heading orientation of the tractor
    local tractorPos = sim.getObjectPosition(tractorBody, -1)
    local currentOrientationEuler = sim.getObjectOrientation(
```

```
        tractorBody, sim.handle_world)
local yaw = sim.alphaBetaGammaToYawPitchRoll(
        currentOrientationEuler[1],
        currentOrientationEuler[2],
        currentOrientationEuler[3])
-- Update current tractor pose with the retrieved data
currentPose.x, currentPose.y, currentPose.heading =
        tractorPos[1], tractorPos[2], yaw

-- Compute distance and angular parameters for navigation
local dx, dy = targetPose.x - currentPose.x,
        targetPose.y - currentPose.y
local distance = math.sqrt(dx^2 + dy^2)
local phi = currentPose.heading - targetPose.phi
local theta = math.atan2(dy, dx) - targetPose.phi
local alpha = math.atan2(math.sin(theta - phi),
        math.cos(theta - phi))

-- Obtain velocity and turning rate measurements from the simulation
local velMes, rateMes = sim.getVelocity(tractorBody)
local omegaZ = rateMes[3]  -- Turning rate around the z-axis
local uMes = velMes[1]*math.cos(currentPose.heading)
```

```
        + velMes[2]*math.sin(currentPose.heading)


-- Non-linear control calculations based on the current state
if math.abs(alpha) < 1e-8 then
    wRef = k*alpha + gamma*math.cos(alpha)*(alpha + h*theta)
    uRef = gamma*math.cos(alpha)*distance
else
    wRef = k*alpha + gamma*math.cos(alpha)
        *math.sin(alpha)/alpha*(alpha + h*theta)
    uRef = gamma*math.cos(alpha)*distance
end


-- Adjust velocities based on control status
if turnRateControl == 0 then wRef = 0 end
if velocityControl == 0 and turnRateControl == 1 then
    angdiff = math.atan2(math.sin(angdiff), math.cos(angdiff))
    wRef = (angdiff > 0 and -0.1) or (angdiff < 0 and 0.1) or 0
    uRef = 0
end


-- Update the graph streams and log data for analysis
sim.setGraphStreamValue(handleGraph, stream1, wRef)
```

```
    sim.setGraphStreamValue(handleGraph, stream2, omegaZ)
    sim.setGraphStreamValue(handleGraph1, stream3, uRef)
    sim.setGraphStreamValue(handleGraph1, stream4, uMes)
    sim.addDrawingObjectItem(trajectory, {tractorPos[1],
            tractorPos[2], 0})

    -- Log position and velocity data to file
    file:write(string.format('time: %.3f [s], wRef: %.3f [rad/s],
            wMes: %.3f [rad/s], uRef: %.3f [m/s], uMes: %.3f [m/s]\n',
            sim.getSimulationTime() + sim.getSimulationTimeStep(),
            wRef, omegaZ, uRef, uMes))
    accuracy:write(string.format("time: %.3f [s], %f, %f\n",
            sim.getSimulationTime() + sim.getSimulationTimeStep(),
            tractorPos[1], tractorPos[2]))
end
```