Autonomous Golf Cart

University of West Florida
Hal Marcus College of Science and Engineering

Erik C. LaBrot
December 1, 2025

Faculty Advisor:
Dr. Tarek Youssef

Presentation Panelists:
Dr. Tarek Youssef
Dr. Minh Ta
Dr. Yazan Alqudah

Abstract

This report details the development of an autonomous driving system implemented on a modified utility golf cart. The system described implements both low level and high level control, realized by a Texas Instruments c2000 series microcontroller, and a Jetson edge computing device. The low level controller interfaces with the golf cart that has been modified in a drive-by-wire style to allow for automated control of steering, throttle, and brake. Likewise, the high level controller interfaces with the c2000 by providing it with velocity and steering angle setpoints, deriving these from a GPS based waypoint follower.

To support controller design, a planar vehicle model is derived that encompasses the longitudinal dynamics and lateral kinematics. This model forms the basis for the controller and simulation design. The simulation is used for controller tuning and algorithm verification before deployment to the physical golfcart.

Contents

List of Figures **5**

List of Tables **6**

1 Introduction **7**
1.1 Motivation and Objectives 7
1.2 Scope 8
1.3 Contributions 8

2 System Overview **10**
2.1 Vehicle Platform 10
2.2 Sensing and Computation 11
2.3 Control Architecture 11

3 Vehicle Hardware Rework **13**
3.1 Initial Vehicle Condition 13
3.2 Control Hardware Update 14
3.3 Electrical Rework 14
3.4 Resulting Platform 15

4 Modeling **16**
4.1 Mechanical Layout and Drivetrain 16
4.2 Simplifications and Assumptions 17
4.2.1 Velocity Operating Region 17
4.2.2 Terrain Considerations 17
4.2.3 Aerodynamics 17
4.2.4 Suspension Dynamics 18
4.2.5 Drivetrain Dynamics 18
4.2.6 Physical Modeling 18
4.3 Longitudinal Dynamics 18
4.4 Lateral Kinematics 19
4.5 Combined Model 21

5 Low-Level Control Design **23**
5.1 Control Objectives and Architecture 23
5.2 Longitudinal Speed Controller Design 24

5.2.1 Drive Motor PI Controller . . . 25
5.2.2 Brake PI Controller . . . 26
5.3 Steering Controller Design . . . 26
5.3.1 Steering PI Controller . . . 27
5.4 Summary . . . 27

6 Low-Level Control Implementation **28**
6.1 Role of the C2000 in the Control Stack . . . 28
6.2 C2000 Peripherals . . . 29
6.2.1 DAC Outputs for Throttle and Brake . . . 29
6.2.2 GPIO for Direction and Brake Control . . . 29
6.2.3 QEP for Speed Measurement . . . 29
6.2.4 SCI for High-Level Communication and Sim Bridge . . . 30
6.3 PI Gain Selection . . . 30
6.3.1 Drive Motor PI Gains . . . 31
6.3.2 Brake PI Gains . . . 31
6.3.3 Steering PI Gains . . . 31
6.4 Simulink Implementation . . . 31
6.4.1 Referenced Subsystems and Harnesses . . . 32
6.4.2 `uart_subsystem` . . . 32
6.4.3 `speed_error_sub` . . . 33
6.4.4 `kls_motor_interface` . . . 34
6.4.5 `brake_subsystem` . . . 35
6.4.6 `openloop_steering_subsystem` . . . 36
6.4.7 `sim_bridge` . . . 38

7 Simulation **40**
7.1 Gazebo Simulator . . . 40
7.2 ROS 2 Middleware and HIL Role . . . 41
7.3 Vehicle Model . . . 41
7.3.1 Link and Joint Structure . . . 41
7.3.2 Sensors . . . 42
7.3.3 Hardware-Consistent Parameters . . . 42
7.3.4 Ackermann Steering Plugin . . . 43
7.4 ros2_control Integration . . . 44
7.4.1 ROS Interface and Parameters . . . 44
7.5 Launch Architecture . . . 44
7.6 C2000 Simulation Bridge . . . 45
7.6.1 Encoder Emulation . . . 46
7.6.2 Serial Protocol and Effort Mapping . . . 46
7.7 Hardware-in-the-Loop Setup . . . 47
7.8 Summary and Limitations . . . 47

8 High Level Control Design **49**
8.1 Jetson Orin Nano . . . 49

8.2 ROS 2 Node Architecture 50
8.3 Implementation Status and Limitations 51

9 Results **53**
9.1 Simulation-Based Controller Validation 53
9.1.1 Longitudinal Speed Controller 54
9.1.2 Brake Controller 55
9.1.3 Steering Angle Controller 57
9.2 Hardware-in-the-Loop and Bench Testing 59
9.2.1 Test Setup 59
9.2.2 Longitudinal Control Results on Hardware 59
9.2.3 Steering Control Results on Hardware 60
9.3 Summary and Limitations 60

10 Conclusion **62**
10.1 Summary of Contributions 62
10.2 Project Outcomes 63
10.3 Closing Remarks 63

A Component specifications **67**

List of Figures

4.1 Mechanical Drivetrain Layout 16
4.2 Steering System Layout 17
4.3 Diagram of Forces Acting on Platform along Longitudinal Axis 19
4.4 Diagram of Bicycle Model 20

5.1 Unity feedback speed control loop with controller $C(s)$ and plant $G(s)$. . . . 25

6.1 `uart_subsystem` for high level command and telemetry 33
6.2 `speed_error_sub` for speed estimation and mode selection 34
6.3 `kls_motor_interface` for KLS throttle, brake, and relay control 36
6.4 `brake_subsystem` implementing the brake PI controller 37
6.5 `openloop_steering_subsystem` for steering command and encoder processing 38
6.6 `sim_bridge` interface between the controller and simulated plant 39

9.1 Results for Step Response to Drive Controller 54
9.2 Results for Step Response to brake Controller 56
9.3 Results for Step Response to brake Controller 58

List of Tables

A.1 Club Car Pioneer 1200 dimensions and weight [1] 67
A.2 Motenergy ME1012 traction motor parameters [2] 68
A.3 Kelly KLS7275D motor controller parameters [3] 68

Chapter 1

Introduction

The University of West Florida autonomous golf cart project is intended to be a platform for autonomous navigation. The vehicle is a utility class electric golf cart controlled with drive-by-wire actuators and an embedded compute unit. This work focuses on the control and software layers that connect those actuators to high level autonomous navigation algorithms.

A TI F28379D C2000 microcontroller executes low level control for traction, braking, and steering. A NVIDIA Jetson Orin Nano runs ROS 2 and hosts higher level control algorithms. A Gazebo-based simulation mirrors the physical system and utilizes the same command and feedback interfaces, so control software can be developed and exercised in simulation before deployment to the vehicle.

1.1 Motivation and Objectives

Autonomous navigation experiments require a control system that behaves in a predictable way and can be reused across different algorithms. Separating low level actuation from high level decision making helps with both. The microcontroller is responsible for tracking speed and steering commands. The Jetson is responsible for deciding which commands to send.

The objectives of this project are to

- derive longitudinal and planar vehicle models suitable for control design and simulation,
- design and implement low level controllers for drive, brake, and steering on the C2000,
- develop a Gazebo-based simulation that uses the same command interface as the physical cart,

- implement ROS 2 nodes for key sensors and for the C2000 link,
- define and begin implementing a high level controller for GPS-based autonomous navigation.

These pieces form the control and software infrastructure needed for later work on path planning, perception, and higher level autonomy.

1.2 Scope

The scope of this report is limited to the control and software that bridge the existing mechanical platform and autonomous navigation. The work includes

- vehicle modeling for longitudinal dynamics and planar motion,
- low level PI control design and embedded implementation on the C2000,
- construction of a Gazebo simulation with a URDF model, Ackermann steering plugin, and ROS 2 bridge,
- ROS 2 integration on the Jetson for GPS, compass, and the C2000 serial interface,
- architecture and partial implementation of a GPS waypoint follower.

This project does not address ride-share scheduling, multi-vehicle coordination, or operation at higher speeds. Perception and obstacle avoidance are only treated at the architectural level. GPS-based waypoint navigation on the physical cart was a target, but only part of the high level functionality was completed.

1.3 Contributions

Within this scope, the contributions of this work to the autonomous golf cart platform are

- a set of longitudinal and planar models tied to measured vehicle parameters and used directly for controller design and simulation,
- a low level controller implementation on the C2000 for traction, braking, and steering, with a unified serial command interface,

- a Gazebo simulation of the cart that shares packet formats and ROS 2 topics with the physical system,
- a ROS 2-based high level control structure on the Jetson that supports GPS-based autonomous navigation and future perception modules.

Chapter 2

System Overview

The autonomous golf cart consists of the physical vehicle, the onboard sensing and compute hardware, and the control software that connects them. The original gasoline Club Car Pioneer 1200 has been rebuilt as an electric, drive-by-wire platform. This chapter summarizes the current system without going into detailed modeling or controller design.

2.1 Vehicle Platform

The base vehicle is a Club Car Pioneer 1200 utility golf cart. In its original form it used an internal combustion engine and conventional driver-operated steering and braking. For this project it has been converted to an electric platform with electronically controlled traction, steering, and braking.

The major physical and drivetrain modifications are:

- The internal combustion engine has been removed and replaced with a 10 kW permanent magnet AC traction motor.
- A 48 V sinusoidal motor controller and supporting power electronics have been installed to drive the traction motor.
- The steering column has been fitted with a mechanical linkage to a DC motor so that steering angle can be commanded electronically.
- A linear actuator has been attached to the brake pedal to apply and release the friction brake under computer control.

These changes allow drive torque, steering angle, and brake force to be commanded by a control system rather than directly by a human driver. The rest of the vehicle (chassis, suspension, and basic mechanical layout) remains close to the original Pioneer 1200 design.

2.2 Sensing and Computation

The platform includes a set of sensors and computation hardware to support autonomous operation.

On the computation side, the current system uses:

- A TI F28379D LaunchPad based on the C2000 architecture as the low level controller. It interfaces with the traction motor controller, brake actuator, throttle pedal, steering motor, and steering encoder, and handles real-time control tasks.
- An NVIDIA Jetson Orin Nano as the high level controller. It runs the navigation and supervisory software and communicates with the low level controller over a defined interface.

On the sensing side, the platform currently uses:

- A SICK LMS200 LiDAR for planar range measurements and obstacle detection in front of the vehicle.
- A GPS receiver for global position estimates used in waypoint navigation.

Additional internal signals such as motor controller status and encoder feedback are also available to the control system through the low level controller. These are used for state estimation, monitoring, and closed loop control.

2.3 Control Architecture

From a control point of view, the system is organized into three layers: a high level control layer, a low level control layer, and the physical platform.

The *high level* control layer runs on the Jetson Orin Nano. Its main job is to control the position of the cart in the environment. It receives GPS and LiDAR data, accepts a sequence of waypoints, plans a path, and generates desired speed and steering commands. This layer

can also host more computationally expensive algorithms such as obstacle avoidance and future mapping or localization methods.

The *low level* control layer runs on the TI F28379D C2000 microcontroller. It takes the desired speed and steering commands from the high level controller and converts them into actuator signals. In practice, it closes feedback loops on vehicle speed and steering angle so that, from the perspective of the high level control, the vehicle behaves like a system with simple setpoints rather than raw motor voltages or currents.

The *platform* layer is the physical hardware: the traction motor and its controller, the steering motor and linkage, the brake actuator, the drivetrain, and the vehicle chassis. This layer determines how commanded torques and forces translate into motion. The interaction between the low level controller and this hardware is the subject of the modeling and controller design discussed in later chapters.

Communication between layers is through well-defined signals. The high level controller sends speed and steering setpoints to the low level controller and receives status and diagnostic information. The low level controller commands the platform actuators and monitors their feedback. This layered structure is intended to keep the system modular and make it easier to extend and maintain.

Chapter 3

Vehicle Hardware Rework

This chapter describes the changes made to the physical platform to support the control and software design in later chapters. The focus is on the control hardware update and the electrical rework that provides a more robust interface between the vehicle, the C2000 microcontroller, and the Jetson.

3.1 Initial Vehicle Condition

When this stage of the project began, the cart had already been converted from its original gasoline drivetrain to an electric traction system. The vehicle carried a traction battery pack, a motor controller, steering and brake actuators, and an assortment of low level control boards.

Power and signal wiring were routed along the frame in plastic cable harnesses. Most connections were made point-to-point. New devices were added by splicing into existing runs and extending the harness. The result worked, but it was not easy to trace individual circuits or change control hardware without disturbing other parts of the system.

For the control work in this report, it was more effective to replace this wiring approach than to continue to extend it. The goal of the rework was to create a layout that is easier to understand, document, and maintain.

3.2 Control Hardware Update

The original low level control used multiple microcontrollers. Several Arduino-class 8-bit boards handled local actuator and sensor tasks, and an ESP32 acted as a coordinator between them and the higher level computer. This distributed arrangement increased the number of devices that had to be configured and maintained.

In the updated design, these boards are replaced by a single TI F28379D C2000-based controller. The C2000 is responsible for:

- generating throttle and brake commands for the traction inverter,
- commanding the steering actuator and reading the steering encoder,
- reading speed feedback from the drive motor encoder,
- exchanging speed and steering setpoints with the Jetson over serial.

Using one controller for all low level tasks simplifies both hardware and software. All actuator interfaces, feedback signals, and control logic reside on a single board. This matches the control structure used in the modeling and low level controller design chapters.

On the high level side, an NVIDIA Jetson Orin Nano serves as the primary compute unit for autonomous navigation. The Jetson connects to the C2000 through a USB–serial link and to sensors such as GPS, compass, and LiDAR through their respective interfaces. The rework ensures that these connections terminate on clear, labeled points instead of ad hoc splices.

3.3 Electrical Rework

The electrical rework centers on moving from embedded harness splices to a panel-based layout using DIN rail and terminal blocks. Power distribution, signal routing, and controller connections are brought into a single point mounted on the vehicle.

The main elements of the new layout are:

- a set of DIN-rail mounted terminal blocks for traction battery, low voltage supply, and signal wiring,
- dedicated terminals for each actuator and sensor connection (traction inverter, steering motor, brake actuator, encoders, GPS, compass, LiDAR),

- separation of high-current power runs from low-voltage control and sensor wiring,
- labeled jumpers and fusing for the 12 V and 5 V control supplies.

The C2000 and Jetson are wired into this panel rather than directly into the vehicle harness. Each I/O line from the controllers appears on a labeled terminal, then routes outward to the corresponding device. This provides a single place to probe signals, insert temporary instrumentation, or reroute connections during testing.

The traction battery feeds a DC–DC converter that supplies the low voltage control bus. From there, regulated rails are distributed through the terminal blocks to the motor controller, actuators, and sensors. This arrangement keeps the power path explicit and makes it clear which devices share a supply.

3.4 Resulting Platform

The hardware rework produces a vehicle that is easier to work with at the control level. The main improvements are:

- low level control consolidated on the C2000,
- a clear electrical interface between the vehicle, the C2000, and the Jetson,
- panel-based power and signal distribution with labeled terminals,
- reduced dependence on in-harness splices and undocumented wiring.

This design supports the modeling, controller design, and implementation work in the following chapters. It also provides future work a defined set of connection points for adding sensors, actuators, or safety systems without repeating another full wiring rework.

Chapter 4

Modeling

This chapter lays out the mathematical framework used in the forthcoming design work. The model derived here is used to dictate simulation modeling as well as controller design. A brief overview of the platform's drivetrain and steering mechanisms is provided, before they are translated to a mathematical expression analytically.

Ultimately, a simplified model based on the bicycle kinematic model is derived, involving the golf cart's longitudinal dynamics to model the relationship between motor torque and velocity.

4.1 Mechanical Layout and Drivetrain

The drivetrain of the golf cart consists of a permanent magnet alternating current (PMAC) motor connected to the original transaxle of the cart. This is accomplished by connecting the motor output to the transaxle input via a chain. From this point, power is transferred to the rear wheels via the transaxle.

The steering system of the golf cart consists of a rack and pinion, controlled by a main steering column to which a DC motor is attached. This connection is mechanically achieved between the output shaft of the motor's gearbox and the steering column via a chain and sprocket. The steering column then actuates the steering rack as if the steering wheel were being turned, according to the DC motor input.

Figure 4.1: Mechanical Drivetrain Layout

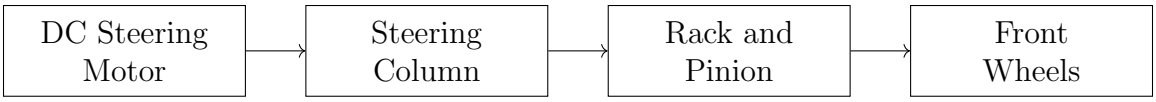

Figure 4.2: Steering System Layout

4.2 Simplifications and Assumptions

The essential idea of the mathematical model for the golf cart is to capture how motor torque relates to linear velocity, and how steering angle changes relate to translation of the platform. The following constraints are imposed to make simulation modeling and controller design more straightforward for this component of the project:

4.2.1 Velocity Operating Region

Due to the low speed operating region of the golf cart (i.e. less than 20 mph), the effects of certain lateral dynamic forces, such as lateral tire slip, are ignored. This simplification is based on Kong et al. [4], whose work shows that below certain speeds and lateral forces, a simplified model is sufficient to describe platform motion. Additionally, a pure rolling model is assumed, where the tires do not slip in the longitudinal direction.

4.2.2 Terrain Considerations

For modeling purposes, terrain is presumed to be level enough such that the inclination or declination of the cart does not meaningfully contribute to the sum of the longitudinal forces acting on the golf cart. Grade forces and load transfer dynamics are excluded from the model. Mathematically, this is represented as any force scaled by $\sin(\theta)$ set to 0.

4.2.3 Aerodynamics

To simplify the modeling math, aerodynamic drag is neglected. At the low speed region the platform operates at, the aerodynamic drag should be small enough to only cause a minor mismatch between simulation and reality.

4.2.4 Suspension Dynamics

Due to the low velocity operating region and terrain assumptions, effects of suspension and pitch dynamics are neglected. It is assumed that the influence these systems would have on the description and control of the vehicle are negligible.

4.2.5 Drivetrain Dynamics

The inertial effect of the drivetrain, such as motor and transaxle inertia, on longitudinal dynamics is neglected for now. Future work may explore experimental measuring of this value to improve the fidelity of the model. However, the inertial effect of the wheel and hub rotation in the sense of effective mass is considered, and is included in the term J_{eq}.

4.2.6 Physical Modeling

For modeling purposes, the cart is assumed to be a symmetric mass with a center of gravity in the geometric center of the body. The effect this has simplifies moments of inertia, and places the CoG equidistant from both axles of the platform.

4.3 Longitudinal Dynamics

The longitudinal dynamics are the body frame forces exerted along the horizontal plane of the platform. These can be derived from Newton's second law, as presented in literature such as Rajamani [5] and Gillespie [6]. Accounting for the assumptions and simplifications made when considering the modeling of the platform, the longitudinal dynamics are as follows:

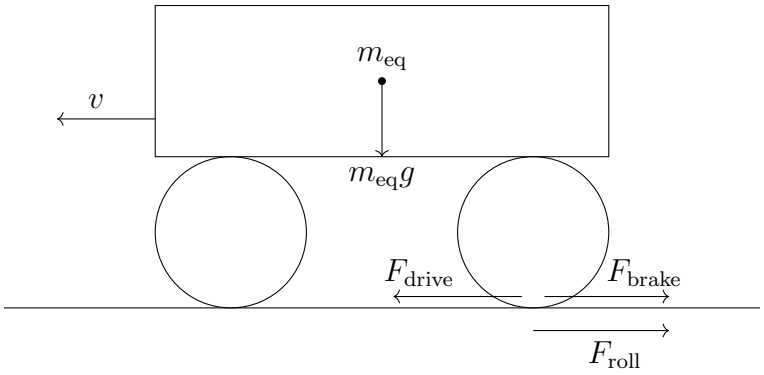

Figure 4.3: Diagram of Forces Acting on Platform along Longitudinal Axis

The above free body diagram is summarized in the following equation:

$$m_{\text{eq}}\,\dot{v} = \sum F_x \tag{4.1}$$

Where the sum of forces is:

$$\sum F_x = F_{\text{drive}} - F_{\text{brake}} - F_{\text{roll}} \tag{4.2}$$

And the individual forces are given by:

$$F_{\text{drive}} = \frac{\eta_d\, i_{\text{tot}}}{r_w}\, T_{\text{motor}} \tag{4.3}$$

$$F_{\text{brake}} = \frac{T_{\text{brake}}}{r_w} \tag{4.4}$$

$$F_{\text{roll}} = C_{rr}\, m\, g \tag{4.5}$$

$$m_{\text{eq}} = m + \frac{J_{\text{eq}}}{r_w^2} \tag{4.6}$$

4.4 Lateral Kinematics

The lateral motion of the golf cart is modeled using a planar kinematic bicycle model. The goal of this model is to capture how changes in steering angle translate to motion of the cart in the world frame. As presented in Kong et al. [4], this simplified representation is sufficient at low speeds and for the smooth maneuvers considered in this work.

A world–fixed frame $\{W\}$ with coordinates (x, y) is defined, and a body-fixed frame $\{B\}$ is attached to the golf cart. The state of the lateral model is

$$\mathbf{q} = \begin{bmatrix} x \\ y \\ \psi \end{bmatrix} \tag{4.7}$$

where (x, y) is the position of a reference point on the cart, the center of the rear axle, expressed in $\{W\}$, and ψ is the yaw heading of the cart with respect to $\{W\}$. The inputs to the kinematic model are the longitudinal speed v along the body x–axis and the front wheel steering angle δ.

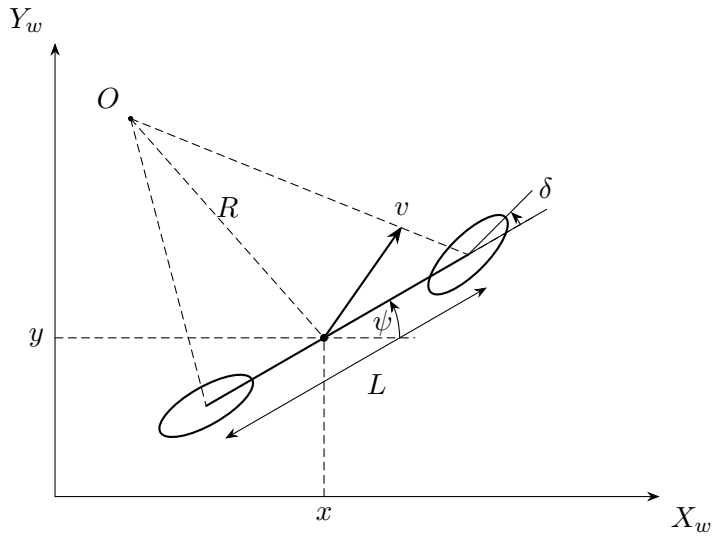

Figure 4.4: Diagram of Bicycle Model

Here, O labels the instantaneous center of rotation, R is the radius of rotation, v is the linear velocity of the cart expressed in $\{W\}$. L is the wheelbase of the platform, and δ is the steering angle expressed in $\{B\}$. Thus, the lateral motion is described by the standard kinematic bicycle equations

$$\dot{x} = v \cos\psi, \tag{4.8}$$

$$\dot{y} = v \sin\psi, \tag{4.9}$$

$$\dot{\psi} = \frac{v}{L} \tan\delta, \tag{4.10}$$

4.5 Combined Model

These two models combined represent the unified set of equations used to model the behavior of the platform. These equations will be used as a basis for designing the parameters of the simulation model, and for designing any necessary controllers. The equations are summarized as the state space model for the platform, and are provided in this section.

Combining the body frame representation (4.7) of the platform with its expression for velocity from the longitudinal dynamics yields the following state vector to describe the motion of the platform with reference to world frame coordinates:

$$\mathbf{x} = \begin{bmatrix} x \\ y \\ \psi \\ v \end{bmatrix} \tag{4.11}$$

The lateral motion is described by the kinematic bicycle model introduced in Section 4.4, with

$$\dot{x} = v \cos \psi, \tag{4.12}$$

$$\dot{y} = v \sin \psi, \tag{4.13}$$

$$\dot{\psi} = \frac{v}{L} \tan \delta, \tag{4.14}$$

where L is the wheelbase and δ is the front wheel steering angle.

The longitudinal dynamics are given by the force balance developed in Section 4.3. Recalling Newton's law in (4.1) with the longitudinal force decomposition in (4.2), the linear acceleration can be described as

$$\dot{v} = \frac{1}{m_{\text{eq}}} \big(F_{\text{drive}} - F_{\text{brake}} - F_{\text{roll}} \big), \tag{4.15}$$

where the effective mass m_{eq} and the individual force terms are defined in (4.3)–(4.5).

For later controller design it is convenient to identify the actuator inputs that enter these equations. The motor torque T_{motor} and brake torque T_{brake} determine the drive and brake forces through (4.3) and (4.4), while the steering angle δ appears in the bicycle kinematics in (4.14). These three control signals can be organized into the following control vector u as

$$\mathbf{u} = \begin{bmatrix} T_{\text{motor}} \\ T_{\text{brake}} \\ \delta \end{bmatrix} \tag{4.16}$$

Using the relationships defined in (4.12)–(4.15), the combined planar model can be expressed in nonlinear state space form as

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v \cos \psi \\ v \sin \psi \\ \frac{v}{L} \tan \delta \\ \frac{\eta_d i_{\text{tot}}}{m_{\text{eq}} r_w} T_{\text{motor}} - \frac{1}{m_{\text{eq}} r_w} T_{\text{brake}} - \frac{C_{rr} m g}{m_{\text{eq}}} \end{bmatrix}. \tag{4.17}$$

Chapter 5

Low-Level Control Design

This chapter presents the low-level controllers used to regulate the golf cart's longitudinal speed and steering angle. The combined planar model in Chapter 4 provides the relationship between motor torque, brake torque, steering angle, and platform motion. That model is used here as the basis for control design. Controls are designed referencing Nise[7] to source general forms for systems and terms.

This control layer maps a speed setpoint $v^*(t)$ and a steering setpoitn $\delta^*(t)$ to actuator commands for the traction motor, friction brake, and steering motor. Three controllers are used:

- a drive motor controller that regulates longitudinal speed using motor torque,
- a brake controller that regulates longitudinal speed using friction brake torque,
- a steering controller that regulates the front wheel steering angle.

5.1 Control Objectives and Architecture

The low-level control layer has two main objectives:

1. Track a commanded longitudinal speed $v^*(t)$ in the operating region $0 \leq v \leq 20$ mph using the traction motor and friction brake.
2. Track a commanded steering angle $\delta^*(t)$ of the front axle, consistent with the kinematic bicycle model in Section 4.4.

A high-level controller generates the setpoints $v^*(t)$ and $\delta^*(t)$ based on its internal error minimization calculations. The low-level controllers operate on the tracking errors

$$e_v(t) = v^*(t) - v(t), \qquad e_\delta(t) = \delta^*(t) - \delta(t), \tag{5.1}$$

and produce commands for the motor controller, brake actuator, and steering drive.

The longitudinal loop behaves similar to a cruise–control style system, with a speed regulator acting on throttle and brake and a higher-level module providing $v^*(t)$. The steering controller acts as a position servo for δ.

The following control-specific assumptions are made:

- The Kelly KLS motor controller and traction motor are modeled as an approximately linear gain between a normalized motor command and an effective drive torque. The brake actuator is modeled in the same way for brake torque.
- The design is restricted to forward motion; behavior very close to zero speed is handled by a stopped/hold mode in implementation.

5.2 Longitudinal Speed Controller Design

The longitudinal dynamics in Section 4.3 and Section 4.5 can be written as

$$\dot{v} = \frac{\eta_d i_{\text{tot}}}{m_{\text{eq}} r_w} T_{\text{motor}} - \frac{1}{m_{\text{eq}} r_w} T_{\text{brake}} - \frac{C_{rr} m g}{m_{\text{eq}}}, \tag{5.2}$$

where T_{motor} and T_{brake} are the effective wheel torques. The rolling resistance term is treated as a constant disturbance for a given operating condition.

Taking inspiration from Ioannou et al,[8], the dynamic model can be simplified based on the domain the error signal exists in. Let $u_{\text{th}} \in [0, 1]$ and $u_{\text{br}} \in [0, 1]$ be normalized throttle and brake commands. In the drive domain (brake released) and the brake domain (throttle released), the dynamics are modeled as

$$\dot{v} = g_d \, u_{\text{th}} - d, \qquad \text{(drive domain)}, \tag{5.3}$$

$$\dot{v} = -g_b \, u_{\text{br}} - d, \qquad \text{(brake domain)}, \tag{5.4}$$

where $g_d > 0$ and $g_b > 0$ are effectively gains and d collects the rolling resistance contribution. The corresponding transfer functions from command to speed (with d neglected in the nominal model) are

$$G_d(s) = \frac{V(s)}{U_{\text{th}}(s)} = \frac{g_d}{s}, \qquad G_b(s) = \frac{V(s)}{U_{\text{br}}(s)} = -\frac{g_b}{s}. \tag{5.5}$$

Domain separation between drive and brake is implemented by a simple deadband on the speed error. Let $e_v = v^* - v$. Then

$$T_{\text{motor}}(t) = \begin{cases} T_{\text{drive}}(t), & e_v(t) > \Delta_v, \\ 0, & e_v(t) \le \Delta_v, \end{cases} \qquad T_{\text{brake}}(t) = \begin{cases} T_{\text{brake,cmd}}(t), & e_v(t) < -\Delta_v, \\ 0, & e_v(t) \ge -\Delta_v, \end{cases} \tag{5.6}$$

where Δ_v is a small deadband threshold, and T_{drive} and $T_{\text{brake,cmd}}$ are proportional to u_{th} and u_{br} through the actuator gains.

5.2.1 Drive Motor PI Controller

In the drive domain, the plant is the integrator

$$G_d(s) = \frac{g_d}{s}. \tag{5.7}$$

The drive controller is a PI compensator acting on the error $e_v = v^* - v$ with transfer function

$$C_v(s) = K_{p,v} + \frac{K_{i,v}}{s}. \tag{5.8}$$

From Nise, with unity feedback, the closed-loop transfer function from v^* to v is

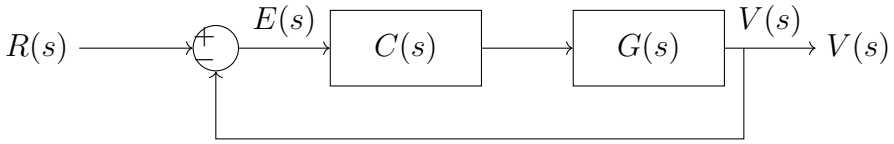

Figure 5.1: Unity feedback speed control loop with controller $C(s)$ and plant $G(s)$.

$$\frac{V(s)}{V^*(s)} = \frac{C_v(s)G_d(s)}{1 + C_v(s)G_d(s)} = \frac{g_d\big(K_{p,v}s + K_{i,v}\big)}{s^2 + g_d K_{p,v} s + g_d K_{i,v}}. \tag{5.9}$$

The denominator in (5.9) defines the closed-loop characteristic polynomial

$$s^2 + g_d K_{p,v}\, s + g_d K_{i,v} = 0. \tag{5.10}$$

To obtain a desired second-order response, (5.10) is matched to

$$s^2 + 2\zeta_v \omega_{n,v} s + \omega_{n,v}^2 = 0, \tag{5.11}$$

where ζ_v is the damping ratio and $\omega_{n,v}$ is the natural frequency of the speed loop. Equating coefficients gives

$$K_{p,v} = \frac{2\zeta_v \omega_{n,v}}{g_d}, \qquad K_{i,v} = \frac{\omega_{n,v}^2}{g_d}. \tag{5.12}$$

Integral action in $C_v(s)$ cancels the steady-state effect of the constant disturbance d in (5.3) for step changes in the speed setpoint.

5.2.2 Brake PI Controller

In the brake domain, the plant is

$$G_b(s) = -\frac{g_b}{s}. \tag{5.13}$$

The brake loop uses a PI controller acting on the error

$$e_b(t) = v(t) - v^*(t), \tag{5.14}$$

which is positive when the vehicle is faster than the setpoint. The controller transfer function is

$$C_b(s) = K_{p,b} + \frac{K_{i,b}}{s}. \tag{5.15}$$

With unity feedback and the sign convention in (5.14), the closed-loop characteristic polynomial again takes the form

$$s^2 + g_b K_{p,b}\, s + g_b K_{i,b} = 0. \tag{5.16}$$

Matching to (5.11) yields

$$K_{p,b} = \frac{2\zeta_b \omega_{n,b}}{g_b}, \qquad K_{i,b} = \frac{\omega_{n,b}^2}{g_b}, \tag{5.17}$$

with $(\zeta_b, \omega_{n,b})$ chosen separately from the drive loop. In practice, braking is tuned to be more heavily damped (larger ζ_b) and sometimes slower (smaller $\omega_{n,b}$) than throttle actuation to improve ride comfort and limit jerk [5, ?].

5.3 Steering Controller Design

The steering controller regulates the front wheel steering angle δ to track a setpoint $\delta^*(t)$ generated by the high-level lateral controller. At the hardware level, the steering column is driven by a DC motor through a motor controller.

The steering mechanism is modeled as a first-order integrator,

$$\dot{\delta}(t) = k_s\, u_\delta(t), \tag{5.18}$$

where u_δ is a normalized steering command and $k_s > 0$ is an effective gain. The corresponding steering plant transfer function is

$$G_s(s) = \frac{\Delta(s)}{U_\delta(s)} = \frac{k_s}{s}, \tag{5.19}$$

where $\Delta(s)$ and $U_\delta(s)$ are the Laplace transforms of $\delta(t)$ and $u_\delta(t)$, respectively.

5.3.1 Steering PI Controller

The steering error is defined as

$$e_\delta(t) = \delta^*(t) - \delta(t), \tag{5.20}$$

and the controller uses a PI structure with transfer function

$$C_\delta(s) = K_{p,\delta} + \frac{K_{i,\delta}}{s}. \tag{5.21}$$

With unity feedback and plant (5.19), the closed-loop transfer function from δ^* to δ is

$$\frac{\Delta(s)}{\Delta^*(s)} = \frac{C_\delta(s)G_s(s)}{1 + C_\delta(s)G_s(s)} = \frac{k_s\big(K_{p,\delta}s + K_{i,\delta}\big)}{s^2 + k_s K_{p,\delta}s + k_s K_{i,\delta}}. \tag{5.22}$$

The denominator defines the closed-loop characteristic polynomial

$$s^2 + k_s K_{p,\delta}\, s + k_s K_{i,\delta} = 0. \tag{5.23}$$

Matching to the standard second-order form (5.11) gives

$$K_{p,\delta} = \frac{2\zeta_\delta \omega_{n,\delta}}{k_s}, \qquad K_{i,\delta} = \frac{\omega_{n,\delta}^2}{k_s}, \tag{5.24}$$

where $(\zeta_\delta, \omega_{n,\delta})$ are the damping ratio and natural frequency of the steering loop.

5.4 Summary

This chapter presented the low-level control design for the golf cart platform using the combined planar model in Chapter 4 as a basis. The equations outlined above describe the systems by which the platform translates its desired high level setpoints into motion of the platform by way of the low level control systems, constructed as standard PI controllers, map measured error to actuator input.

Chapter 6

Low-Level Control Implementation

This chapter describes how the low level controllers run on the F28379D C2000 microcontroller, how the PI gains are chosen from a target settling time using the design equations from Section 5, and how the Simulink model is organized so the same controller logic runs in simulation and on hardware

6.1 Role of the C2000 in the Control Stack

The C2000 board closes the low level loops for speed and steering, and interfaces with the high level controller. It is situated between the high level computer and the power electronics for the traction motor and steering actuator In this project the C2000 is responsible for:

- Receives target longitudinal speed and target steering angle from the high level controller
- Measures vehicle speed and steering angle from onboard sensors
- Runs three PI controllers: throttle, brake, and steering
- Outputs analog throttle and brake commands and a PWM style steering command
- Reports basic telemetry and fault status back to the high level controller

6.2 C2000 Peripherals

The low level controller runs on the F28379D and uses four main peripheral groups. These peripherals provide the physical interfaces to the traction motor controller, brake actuator, steering hardware, and the high level computer.

6.2.1 DAC Outputs for Throttle and Brake

Throttle and brake commands are analog voltages. The LaunchXL-F28379D provides DAC outputs implemented internally using ePWM channels and an on board low pass filter.

Each longitudinal PI controller produces a normalized command in $[0, 1]$ This value is written to a DAC block, which generates a filtered output is a 0 V to 3 V signal at the header.

One DAC channel drives the Kelly KLS-D motor controller pedal input through an external 0 V to 5 V scaling circuit. A second DAC channel drives the brake actuator interface with a 0 V to 5 V scaling circuit. This gives independent analog outputs for the throttle and brake PI controllers.

6.2.2 GPIO for Direction and Brake Control

Digital GPIO pins provide the discrete control signals required by the Kelly KLS-D. These pins are driven directly from logic in the low level controller.

- Forward and reverse direction selection for the KLS
- Parking engine brake for KLS
- Throttle signal source selection

6.2.3 QEP for Speed Measurement

Wheel speed is measured using a quadrature encoder and a C2000 QEP unit. The QEP hardware counts encoder edges and exposes a position count that is read once per control step. This hardware count is then differentiated over control steps to measure wheel velocity, or compared directly to an initial value to find steering angle.

6.2.4 SCI for High-Level Communication and Sim Bridge

Serial Communication Interface (SCI) ports connect the C2000 to the rest of the system. The same peripheral family is used for three roles:

- Command and telemetry link to the high level compute
- Packetized serial link to the steering motor controller
- Sim bridge connection for hardware-in-the-loop operation with a software plant

Over these links the low level controller receives target speed v^*, target steering angle δ^*, mode and enable commands, and returns measured speed v, steering angle δ, and controller status and fault flags. One of these links is dedicated to interfacing with the steering motor controller using packetized serial mode.

6.3 PI Gain Selection

The PI gains for all low level loops are taken directly from the design result in Chapter 5 For an integrator plant

$$G_x(s) = \frac{g_x}{s} \tag{6.1}$$

with a PI compensator

$$C_x(s) = K_{p,x} + \frac{K_{i,x}}{s}, \tag{6.2}$$

the derivation in Section 5.2.1 shows that matching the closed loop characteristic polynomial to a standard second order form gives

$$K_{p,x} = \frac{2\zeta_x \omega_{n,x}}{g_x}, \qquad K_{i,x} = \frac{\omega_{n,x}^2}{g_x}. \tag{6.3}$$

For each loop x the only design choice is the damping ratio ζ_x and natural frequency $\omega_{n,x}$ The natural frequency is expressed in terms of a target 2% settling time $t_{s,x}$ using the standard second order approximation

$$t_{s,x} \approx \frac{4}{\zeta_x \omega_{n,x}} \quad \Rightarrow \quad \omega_{n,x} = \frac{4}{\zeta_x t_{s,x}}. \tag{6.4}$$

Given ζ_x and $t_{s,x}$, $\omega_{n,x}$ is computed from (6.4) and the gains follow directly from (6.3)

6.3.1 Drive Motor PI Gains

In the drive domain the plant gain is $g_x = g_d$ as defined in Chapter 5 The speed loop uses a damping ratio ζ_v and settling time $t_{s,v}$ The resulting gains are

$$K_{p,v} = \frac{2\zeta_v \omega_{n,v}}{g_d}, \qquad K_{i,v} = \frac{\omega_{n,v}^2}{g_d}, \tag{6.5}$$

with $\omega_{n,v}$ obtained from (6.4) using $t_{s,v}$ and ζ_v

6.3.2 Brake PI Gains

The brake loop uses the same structure with plant gain g_b A damping ratio ζ_b and settling time $t_{s,b}$ are chosen for the braking response The gains are

$$K_{p,b} = \frac{2\zeta_b \omega_{n,b}}{g_b}, \qquad K_{i,b} = \frac{\omega_{n,b}^2}{g_b}, \tag{6.6}$$

with $\omega_{n,b}$ computed from (6.4)

6.3.3 Steering PI Gains

The steering loop is modeled as an integrator with gain g_δ Using a damping ratio ζ_δ and target steering settling time $t_{s,\delta}$, the steering PI gains are

$$K_{p,\delta} = \frac{2\zeta_\delta \omega_{n,\delta}}{g_\delta}, \qquad K_{i,\delta} = \frac{\omega_{n,\delta}^2}{g_\delta}, \tag{6.7}$$

again with $\omega_{n,\delta}$ computed from (6.4)

In the implementation all three loops use the same nominal damping ratio, $\zeta_v = \zeta_b = \zeta_\delta = 0.7$, so the primary tuning knobs are the settling times $t_{s,v}$, $t_{s,b}$, and $t_{s,\delta}$ Once the gains g_d, g_b, and g_δ are fixed from the Chapter 5 models, the PI gains follow directly from the expressions above

6.4 Simulink Implementation

The implementation method is programming the F28379D using the embedded coder workflow combined with the C2000 toolbox for Matlab and Simulink. This allows for the c2000

family of microcontrollers to be programmed using Simulink system diagrams. The advantage is that systems can be represented graphically before being converted into code, which presents as a more readable and understandable format.

Each major function is implemented as a referenced subsystem. Referenced subsystems allow for functionality to be contained in an isolated module, where logic, inputs, and outputs can be defined and then used elsewhere. The design pattern is similar to functional programming in that logic and implementation are kept seperate to allow for clarity and maintainability. The subsystem modules are listed, then explored in detail in this section.

6.4.1 Referenced Subsystems and Harnesses

At the top level, the simulation harness contains the following subsystems:

- `uart_subsystem` – command and telemetry link to the high level computer.
- `speed_error_sub` – speed estimation, error generation, and drive/brake mode selection.
- `kls_motor_interface` – mapping of longitudinal control signals to KLS throttle, brake, and relay outputs.
- `brake_subsystem` – brake PI controller and brake command generation.
- `openloop_steering_subsystem` – steering command and steering encoder emulation.
- `sim_bridge` – interface between the controller and the simulated plant.

The platform harness instantiates the same set of subsystems, but connects their ports to C2000 peripheral blocks instead of the simulation bridge.

6.4.2 `uart_subsystem`

The `uart_subsystem` handles command and telemetry exchange with the high level computer over SCI.

On the receive side, it:

- Unpacks incoming bytes into motor throttle command, direction and brake flags, and steering setpoint.
- Outputs these values as `MOTOR_THR_CMD`, `MOTOR_THR_DIR`, `MOTOR_THR_BRK`, and `SHAFT_ANGLE_CMD`.

On the transmit side, it:

- Receives `MOTOR_RPM` and `SHAFT_ANGLE` from the rest of the model.
- Packs these signals into a byte stream and transmits them back to the computer.

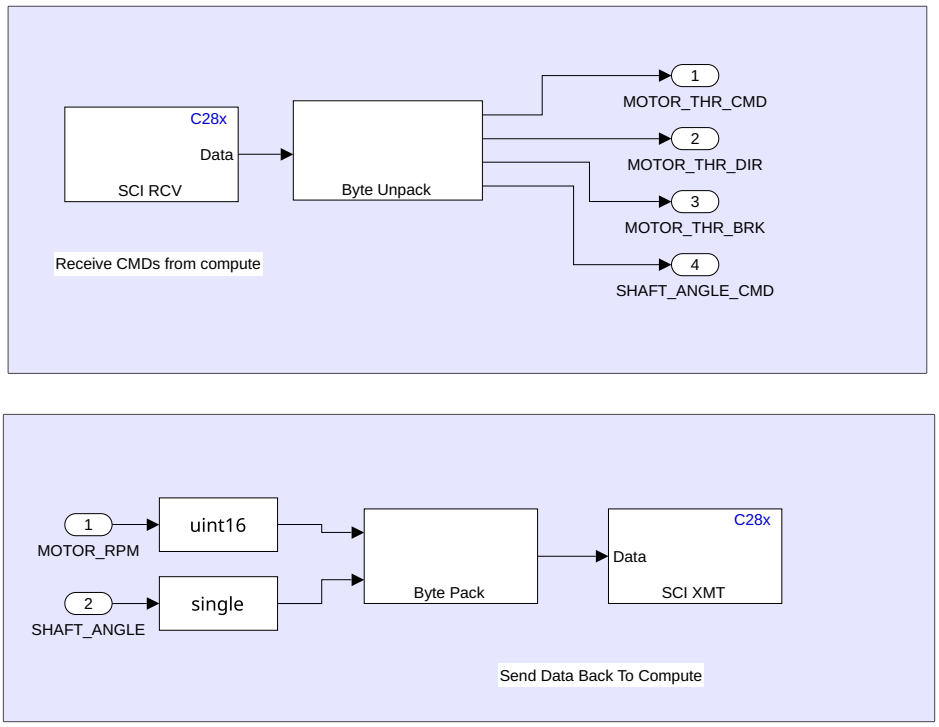

Figure 6.1: `uart_subsystem` for high level command and telemetry

6.4.3 `speed_error_sub`

The `speed_error_sub` block implements the longitudinal speed estimation and error logic that is shared by the drive and brake paths.

It:

- Receives the target speed from the computer as `TARGET_VEL`.

- Receives simulated encoder counts from the plant as `DRIVE_ENC`.
- Converts encoder counts to motor speed `MOTOR_RPM`.
- Computes the velocity error `VEL_ERR` between commanded and measured speed.
- Generates enable flags `DRIVE_EN` and `BRK_EN` based on the sign and magnitude of the error.

The `VEL_ERR` signal is fed to both the `kls_motor_interface` and the `brake_subsystem`. `DRIVE_EN` and `BRK_EN` gate the corresponding loops so that only one of drive or brake is active at a time. `MOTOR_RPM` is returned to the `uart_subsystem` for telemetry.

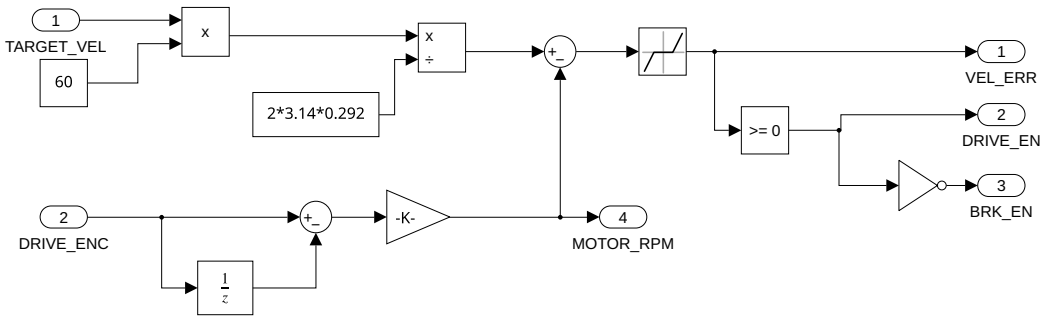

Figure 6.2: `speed_error_sub` for speed estimation and mode selection

6.4.4 `kls_motor_interface`

The `kls_motor_interface` converts the longitudinal control signals into commands compatible with the Kelly KLS motor controller.

Inputs to this subsystem are:

- THROTTLE_BRK – combined longitudinal command (in simulation, this is driven from the brake and drive logic).
- VEL_ERROR – velocity error from speed_error_sub.
- DRIVE_EN – enable signal for the drive PI loop.

The subsystem:

- Applies the drive PI controller to VEL_ERROR when DRIVE_EN is asserted.
- Maps the PI output to a DAC command for the KLS pedal input.
- Drives GPIO outputs for FWD_REL, REV_REL, and BRK_REL to select direction and braking mode.
- Produces SIM_VEL_CMD, a normalized drive command used by the simulation bridge in the simulation harness.

In the platform harness, DAC_CMD and relay outputs connect directly to C2000 DAC and GPIO blocks. In the simulation harness, SIM_VEL_CMD is routed to the sim_bridge and the hardware-specific outputs are left unconnected or used for monitoring.

6.4.5 brake_subsystem

The brake_subsystem contains the brake PI controller and generates the brake command used in both hardware and simulation.

It receives:

- OUTPUT_EN – brake enable signal, driven from BRK_EN.
- VEL_ERROR – velocity error from speed_error_sub.

When OUTPUT_EN is high, the subsystem:

- Applies the brake PI controller to VEL_ERROR using the gains from Section 6.3.

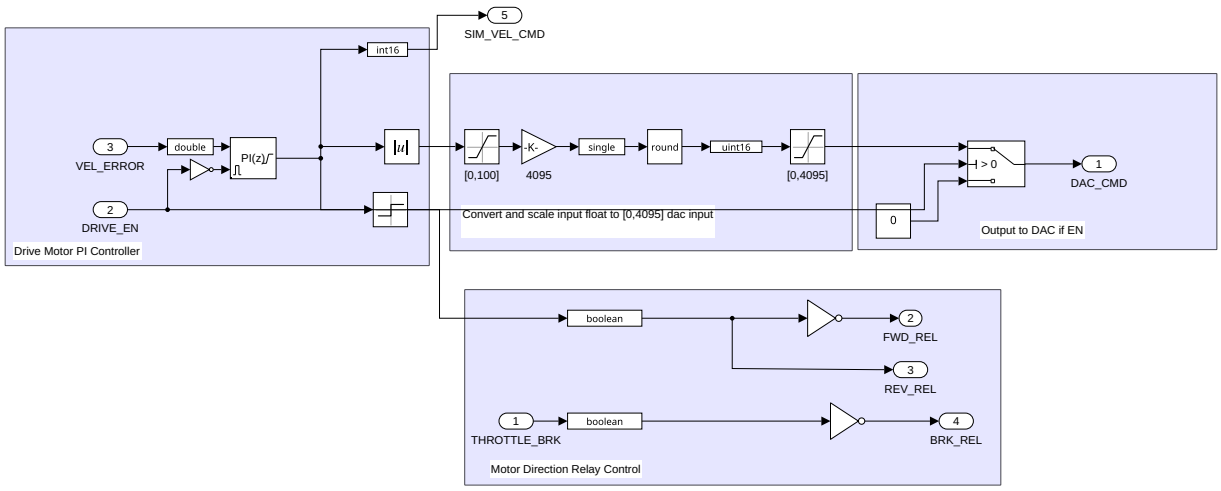

Figure 6.3: `kls_motor_interface` for KLS throttle, brake, and relay control

- Generates a DAC level `DAC_CMD` for the brake actuator interface.
- Generates a normalized brake command `SIM_BRK_CMD` for the simulated plant.

In the platform harness, `DAC_CMD` is routed to a C2000 DAC channel that drives the physical brake actuator. In the simulation harness, `SIM_BRK_CMD` is connected to the `sim_bridge`.

6.4.6 `openloop_steering_subsystem`

The `openloop_steering_subsystem` handles steering commands and steering encoder emulation.

Inputs are:

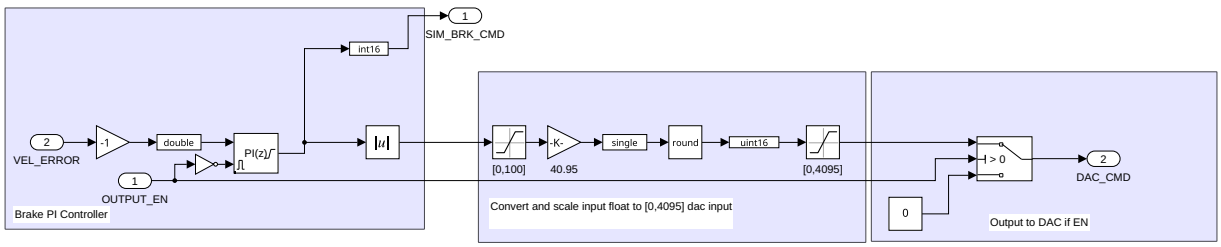

Figure 6.4: `brake_subsystem` implementing the brake PI controller

- `STEER_EN` – steering enable signal.
- `STEER_SETPOINT` – desired steering angle from `SHAFT_ANGLE_CMD`.
- `ENC_INP` – steering encoder feedback from the simulated plant or hardware.

The subsystem:

- Shapes and limits the steering command based on `STEER_SETPOINT` and `STEER_EN`.
- Outputs a normalized steering command `STEER_CMD` to the steering motor driver (platform) or `sim_bridge` (simulation).
- Processes encoder feedback to produce `STEER_ENC_CNT` and `STEER_ENC_ROT` signals.
- Returns a shaft angle estimate to the `uart_subsystem` for telemetry.

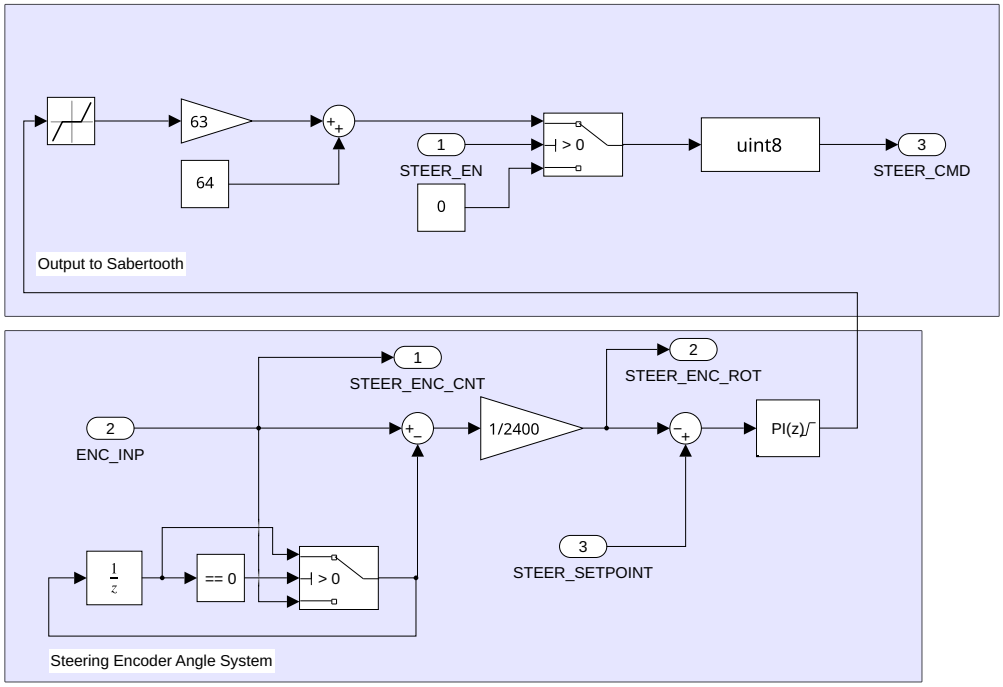

Figure 6.5: `openloop_steering_subsystem` for steering command and encoder processing

6.4.7 `sim_bridge`

The `sim_bridge` connects the controller to the software plant during simulation.

On the command side it receives:

- `THROTTLE_CMD` from `kls_motor_interface` (`SIM_VEL_CMD`).
- `BRAKE_CMD` from `brake_subsystem` (`SIM_BRK_CMD`).
- `STEER_CMD` from `openloop_steering_subsystem`.

On the feedback side it outputs:

- `DRIVE_ENC` – simulated drive encoder counts to `speed_error_sub`.

- `STEER_ENC` – simulated steering encoder feedback to `openloop_steering_subsystem`.
- `SIM_CONNECTED` – a status flag used to gate commands when the plant connection is not valid.

Internally, the `sim_bridge` handles packetization and transport over the serial or socket interface to the host simulation. The controller model itself remains unaware of whether the plant is simulated or physical; it only sees encoder feedback and status flags at its ports.

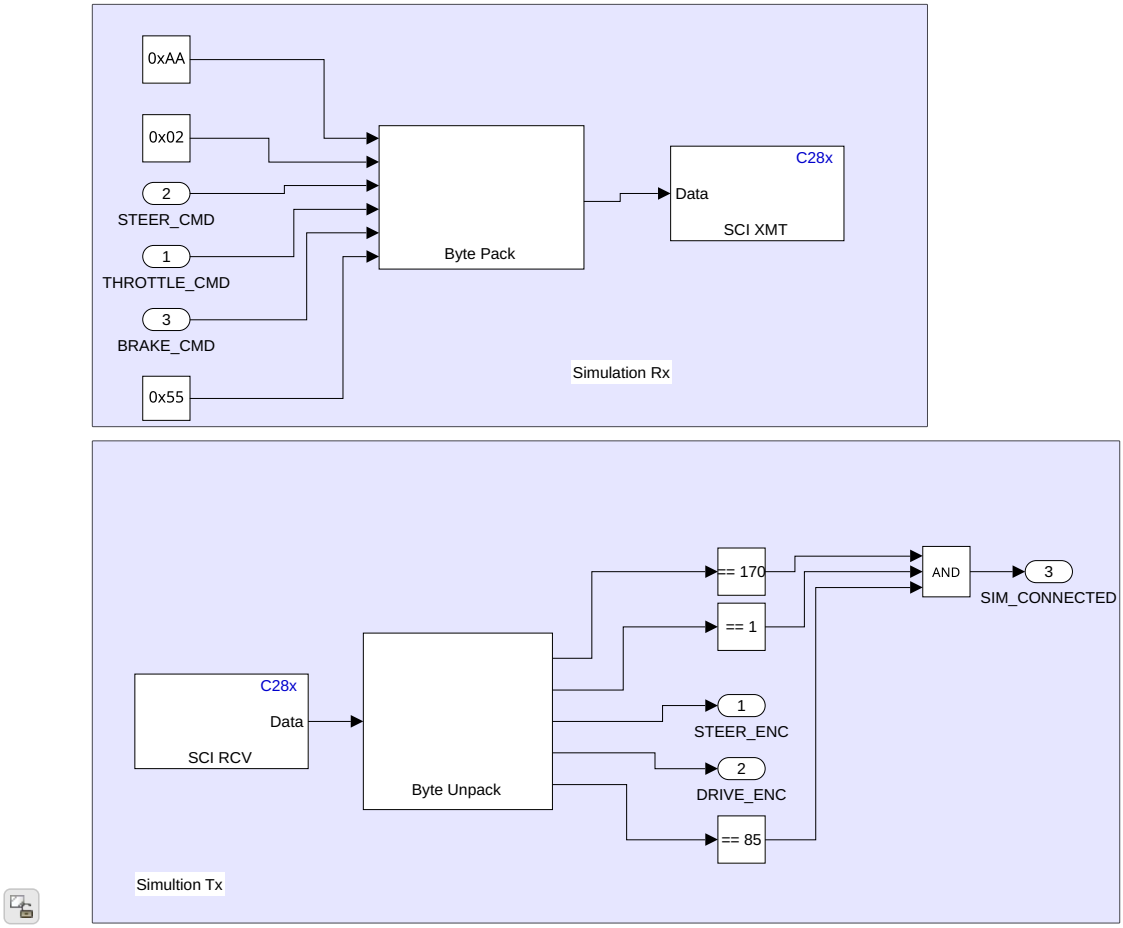

Figure 6.6: `sim_bridge` interface between the controller and simulated plant

Chapter 7

Simulation

This chapter describes the Gazebo-based simulation and ROS 2 integration used to exercise the low level controllers and the C2000 implementation before testing on the physical vehicle. The goal is to replace the golf cart with a simulated analogue that exposes the same command and feedback interfaces as the real platform. Chapter 2 gives a high level view of the system, while this chapter focuses on the simulation architecture and its role in the control stack.

7.1 Gazebo Simulator

Gazebo Sim is an open-source robotics simulator that provides rigid body dynamics, contact modeling, and sensor plugins suitable for robot development [9]. In this project it serves as the plant model for the golf cart.

The `uwf_empty.world` file defines a deliberately minimal environment:

- A flat ground plane of size $100 \times 100\,\mathrm{m}$,
- A physics configuration with a 1 ms maximum step size and a real-time factor near 1.0,
- The standard Gazebo physics system plugin and default gravity.

The world is kept feature-light so that changes in vehicle behavior can be traced to the controller or model configuration. Obstacles, terrain variation, and additional scenery can be added later when moving from controller bring-up to higher level navigation experiments.

7.2 ROS 2 Middleware and HIL Role

ROS 2 is an open-source middleware framework for robotics that provides message passing, parameter management, and lifecycle tools over a DDS transport [10]. It is widely used as the integration layer in modern robotic platforms, and it plays the same role here.

In this project ROS 2 sits between:

- Gazebo, which simulates the vehicle dynamics and sensors,
- The C2000 controller, running Simulink-generated low level firmware,
- Auxiliary nodes such as controller managers and bridge nodes.

Within ROS 2, several subsystems are central to the simulation:

- `ros2_control`, which provides a standardized interface for joint-level controllers and hardware abstraction [11],
- `ros_gz_sim` and `ros_gz_bridge`, which connect Gazebo entities (joints, links, and sensors) to ROS 2 topics and services [12].

Together these components allow the C2000 firmware to interact with the simulated vehicle using the same command and feedback channels that it will use on the physical cart. This makes the simulation a hardware-in-the-loop (HIL) testbed once the real microcontroller is connected.

7.3 Vehicle Model

The golf cart is modeled as a URDF/Xacro description in the `agc_golfcart_sim` package. The URDF specifies the kinematic tree, masses, inertias, joint limits, and collision geometries used by Gazebo. The parameters are chosen to be physically meaningful and consistent with the modeling and hardware chapters, rather than arbitrary values.

7.3.1 Link and Joint Structure

The root link is `base_footprint`, kept for compatibility with ROS navigation and visualization tools. A fixed transform links `base_footprint` to the `chassis` link, which carries the lumped mass and inertia of the vehicle body:

- Mass on the order of 500 kg, matching the Pioneer 1200 curb mass with the new powertrain installed,
- Diagonal inertia entries chosen to approximate the mass distribution about the center of gravity based on manual data and measurements.

The remaining links and joints are organized as a tree:

- Four wheel links (`front_left_wheel`, `front_right_wheel`, `rear_left_wheel`, `rear_right_wheel`),
- Revolute wheel joints about the y-axis to allow rolling motion,
- A virtual `steering_input_joint` representing the steering column input,
- Physical steering knuckles (`front_left_steering_joint`, `front_right_steering_joint`) which rotate the front wheels.

All joints include limits, damping, and friction for numerical stability and to approximate real hardware behavior.

7.3.2 Sensors

Three simulated sensors are mounted on the chassis:

- A planar LiDAR on `lidar_link` publishing a `sensor_msgs/LaserScan` on `/lidar`,
- A GPS sensor on `gps_link` publishing a `sensor_msgs/NavSatFix` on `/gps`,
- An IMU on `imu_link` publishing a `sensor_msgs/Imu` on `/imu`.

The LiDAR provides 2D range data for localization and obstacle detection in later work. The GPS and IMU provide global position and inertial data consistent with the physical sensor suite on the cart. Noise models are left modest so that controller behavior remains readable during tuning, but the interfaces match those of real sensors.

7.3.3 Hardware-Consistent Parameters

The URDF parameters are chosen to mimic the physical hardware:

- **Vehicle dimensions:** Wheelbase, front and rear track widths, and overall length are set from the Pioneer 1200 manual and verified by direct measurements on the modified cart.
- **Wheel and tire geometry:** Wheel radius and tire width are taken from the installed tire size so that the mapping between wheel angular velocity and vehicle speed is consistent.
- **Mass and inertia:** Curb mass is based on manufacturer data with adjustments for removed ICE components and added electric components. Inertias are tuned within a narrow range to keep simulations stable while remaining physically plausible.
- **Steering geometry:** Wheelbase and front track width match the values used in the kinematic bicycle model in Chapter 4, so the relationship between steering input and turn radius is consistent.
- **Drivetrain limits:** Maximum drive torque, brake torque, and steering effort are based on motor and actuator datasheets, with slight reductions to keep the simulation numerically stable and conservative.

These choices ensure that step responses in simulation occupy the same order of magnitude as the hardware and stay within the operating range assumed in the modeling and control design chapters.

7.3.4 Ackermann Steering Plugin

The front steering geometry is handled by a custom Gazebo system plugin implemented in `plugins/gazebo/ackermann_steering/ackermann_steering_plugin.cpp` and loaded as `libackermann_steering_plugin.so`. The plugin implements standard Ackermann steering relations between a bicycle-model steering angle and the individual front wheel angles [13].

The controller commands a single virtual steering angle δ on `steering_input_joint`, corresponding to the front axle angle in the kinematic bicycle model. Given wheelbase L and front track width w, the corresponding turn radius R of the vehicle path is

$$R = \frac{L}{\tan \delta}. \tag{7.1}$$

For this turn radius, the inner and outer wheel steer angles δ_{in} and δ_{out} that approximately satisfy Ackermann steering geometry are

$$\delta_{\text{in}} = \arctan\left(\frac{L}{R - \frac{w}{2}}\right), \tag{7.2}$$

$$\delta_{\text{out}} = \arctan\left(\frac{L}{R + \frac{w}{2}}\right). \tag{7.3}$$

The plugin computes these wheel angles from the commanded δ and writes them to the `front_left_steering_joint` and `front_right_steering_joint`. This keeps the simulated kinematics consistent with the bicycle model used in controller design while still producing realistic front wheel motion in Gazebo.

7.4 ros2_control Integration

The interface between ROS 2 and Gazebo is built around `ros2_control`. The golf cart model includes a `ros2_control` plugin attached to the chassis link, with joint interfaces defined for steering and rear drive.

7.4.1 ROS Interface and Parameters

Controller configuration in `config/ros2_controllers.yaml` declares:

- A steering controller that commands the `steering_input_joint` in position (radians),
- A rear drive controller that commands the driven wheel joints in effort (Newton-meters),
- Joint state interfaces so that all joint positions, velocities, and efforts are published on `/joint_states`.

The steering controller accepts target steering angles from the high level controller. The rear drive controller accepts commanded drive efforts that represent the combined effect of throttle and brake after the low level PI controllers. These controllers are standard `ros2_control` controllers and are managed by the `controller_manager` node [11].

7.5 Launch Architecture

The main simulation launch file `launch/simulation.launch.py` orchestrates:

1. Starting Gazebo with `uwf_empty.world`,
2. Generating the URDF at runtime by running `xacro` on `model.xacro.urdf`,
3. Starting `robot_state_publisher` with the generated `robot_description`,

4. Spawning the golf cart model into Gazebo using the published description,
5. Starting the ROS–Gazebo bridge to connect Gazebo topics (joints, sensors) to ROS 2 topics [12],
6. Spawning the `ros2_control` controllers,
7. Launching the C2000 simulation bridge node.

Environment variables are set so Gazebo can locate the model and custom plugins:

- `IGN_GAZEBO_RESOURCE_PATH` includes the package `models` directory,
- `AGC_CONFIG_PATH` points to the `config` directory so that YAML files can be found by both launch files and nodes.

This launch structure mirrors how the real system will be brought up: Gazebo takes the role of the physical vehicle and environment, while the remaining nodes are the same ones used in bench and field tests.

7.6 C2000 Simulation Bridge

The C2000 simulation bridge is a ROS 2 node that emulates the wheel encoder and steering sensor signals seen by the C2000 and receives motor and steering commands from it. It allows the same Simulink-generated firmware to run untouched, whether the plant is Gazebo or the real golf cart.

The node is implemented in Python in the `agc_c2000_interface` package. It is configured via `config/c2000_bridge_params.yaml`, which sets:

- Serial device name, baud rate, and update rate,
- Joint names for steering and rear drive,
- Encoder counts per revolution and steering encoder center value,
- Steering gear ratio mapping joint radians to steering column radians,
- Scaling factors between C2000 command bytes and ROS 2 effort commands.

7.6.1 Encoder Emulation

The bridge converts simulated joint states into QEP-style encoder counts expected by the C2000 firmware:

- It subscribes to `/joint_states`,
- It extracts the steering joint angle and rear wheel joint angle,
- It maintains integer encoder counts for steering and drive based on configured counts-per-revolution and gear ratio.

For steering, the node tracks the last steering angle, converts angle differences into counts using the configured counts-per-revolution and steering gear ratio, and updates the steering encoder value. For the drive wheels, it integrates the rear wheel rotation into counts so that the C2000 can compute wheel speed from successive differences, just as it does with a real QEP. This makes the C2000 firmware unaware that Gazebo is providing the "sensor" data.

7.6.2 Serial Protocol and Effort Mapping

The bridge implements the same serial packet structure used between the C2000 and the physical interface board. Two packet types are used:

- Sensor packets from bridge to C2000, carrying steering and drive encoder counts and status bits,
- Motor command packets from C2000 to bridge, carrying throttle, brake, and steering command bytes.

On receiving a command packet, the bridge:

1. Decodes the throttle and brake command bytes and maps them to a net rear axle effort,
2. Decodes the steering command byte and maps it to a desired steering angle at the steering column,
3. Converts the steering column angle to a `steering_input_joint` target using the steering gear ratio,
4. Publishes the resulting commands to the steering and rear drive controllers as effort or position targets,

5. Packages the latest encoder counts into a sensor packet and transmits it back to the C2000 at a fixed rate.

Because the C2000 sees the same packet structure, scaling, and timing in simulation and on hardware, the same firmware image can be used in both cases.

7.7 Hardware-in-the-Loop Setup

When the real C2000 board is connected over USB serial to the simulation host, the system forms a HIL loop:

- The C2000 runs the low level PI controllers described in Chapter 5 and implemented in Chapter 6,
- The C2000 simulation bridge provides encoder counts and receives command packets,
- `ros2_control` translates the commands into joint efforts and positions,
- Gazebo advances the vehicle dynamics and publishes joint states and sensor data,
- The bridge converts joint states back into encoder counts and the cycle repeats.

This setup allows PI gains, limiters, and safety logic to be tuned against a dynamic plant without running the real traction motor or moving the vehicle. Only the serial connection and supply power for the C2000 are required.

7.8 Summary and Limitations

This simulation stack:

- Provides a dynamic model consistent with the modeling assumptions in Chapter 4,
- Mimics the physical hardware using datasheet-based parameters and measured dimensions,
- Reproduces the C2000 encoder and serial interfaces closely enough to reuse the same Simulink firmware in simulation and on hardware,
- Uses ROS 2 as the integration layer between the Gazebo plant and the C2000 controller, enabling full HIL testing.

Several limitations remain:

- The terrain is flat and rigid, with no suspension compliance or body roll effects,
- Tire forces are modeled using simple friction and damping, not a full tire model with slip and combined loading,
- Sensor models include only modest noise and latency and do not yet capture all real-world artifacts,
- Power electronics, thermal limits, and battery dynamics are not modeled in detail.

These limitations are acceptable for validating the low level controllers and integration. They should be revisited when moving to more aggressive maneuvers, rough terrain, or when closing higher level perception and planning loops on the real golf cart.

Chapter 8

High Level Control Design

This chapter describes the high level control layer that runs on the Jetson edge computer. The goal of this layer is to turn global navigation objectives into local speed and steering setpoints for the low level controller described in Chapter 5. It receives GPS and heading measurements, manages a list of waypoints, and generates commands for the C2000 microcontroller over a serial link.

The design presented here reflects the intended architecture for GPS waypoint following. The supporting ROS 2 nodes for GPS, compass, and C2000 communication are implemented and tested, but the full waypoint following controller has not yet been deployed on the physical cart.

8.1 Jetson Orin Nano

The high level controller runs on an NVIDIA Jetson Orin Nano mounted on the vehicle. This module provides a multi-core CPU and GPU, along with UART, I^2C, and USB interfaces for sensor and actuator connections. It serves as the main compute platform for navigation, perception, and supervisory control.

On the hardware side, the Jetson connects to:

- The Ublox GPS receiver over a UART port exposed as `/dev/ttyTHS1`,
- The HMC5883L magnetometer over the I^2C bus,
- The SICK LMS200 LiDAR through an external interface (used primarily for future obstacle detection),

- The C2000 microcontroller over a USB–serial adapter,
- An Ethernet or Wi-Fi link for development and monitoring.

The Jetson runs a Linux operating system and hosts a ROS 2 stack inside a containerized environment. This mirrors the ROS 2 installation used on the development workstation so that the same launches and nodes can be exercised both in simulation and on hardware. ROS 2 provides the node, topic, and parameter abstractions used to organize the high level control software [10].

From a control perspective, the Jetson exposes a single upstream interface to the rest of the system: it publishes desired longitudinal speed $v^*(t)$ and steering angle $\delta^*(t)$ as ROS 2 messages. Everything else inside the high level layer (sensor drivers, waypoint management, and control logic) is hidden behind this interface.

8.2 ROS 2 Node Architecture

High level control on the Jetson is organized as a small set of ROS 2 nodes. Each node has a narrow, well-defined responsibility and communicates using standard ROS message types.

At the sensor interface, the `agc_sensors` package provides a node `gps_compass_node`. This node:

- Opens the GPS serial port with a configurable baud rate,
- Parses UBX navigation messages into latitude, longitude, and altitude,
- Publishes GPS fixes as `sensor_msgs/NavSatFix` on the `gps/fix` topic,
- Reads heading from the HMC5883L magnetometer over I^2C,
- Publishes heading in degrees as `std_msgs/Float64` on `compass/headingDeg`.

This node encapsulates all device-specific handling for the GPS and compass. Downstream controllers can treat its outputs as generic position and heading measurements without needing to be aware of message formats or low level protocols.

At the actuator interface, the `agc_c2000_interface` package provides the `c2000_tx` node. This node:

- Subscribes to an `agc_msgs/C2000Command` message on the `c2000/cmd` topic,

- Packs the command into a compact byte frame consisting of: a throttle magnitude byte, direction and brake flags, and a floating point steering angle,
- Sends the packed frame over a serial port to the C2000 microcontroller,
- Supports either event-driven transmission (write-on-change) or periodic transmission at a fixed rate.

The C2000 receives this frame, unpacks it in Simulink, and uses the contents as setpoints for the low level PI controllers. This keeps the ROS 2 side focused on high level quantities (speed and steering) while the microcontroller manages all hardware details.

Between these two ends sits the waypoint follower node, which is responsible for:

- Maintaining the list of GPS waypoints that define the route,
- Converting the current GPS fix into a local planar coordinate frame,
- Identifying the current target segment of the route,
- Computing a desired speed $v^*(t)$ and steering angle $\delta^*(t)$ based on the cross-track and heading error to that segment,
- Publishing the corresponding `C2000Command` messages on `c2000/cmd`.

In the current project state, the GPS/compass and C2000 interface nodes are implemented and have been exercised on the Jetson. The waypoint follower node is specified and partially prototyped, but was not completed to a state where full autonomous waypoint following could be demonstrated on the physical cart.

8.3 Implementation Status and Limitations

While the ROS 2 infrastructure for high level control is largely in place, the full GPS waypoint following behavior was not completed within the project timeline. Specifically:

- The `gps_compass_node` has been tested on the Jetson and verified to publish reasonable GPS fixes and compass headings.
- The `c2000_tx` node has been verified to send correctly formatted command frames to the C2000 over serial.

As a result, the high level control design in this chapter should be viewed as an implemented sensor and actuator interface rather than a complete autonomous driving stack. Finishing the waypoint follower and integrating it with the existing low level controllers is the next step for the autonomous golf cart project.

Chapter 9

Results

This chapter summarizes the validation of the models, controllers, and software architecture developed in the previous chapters. Results are organized into two main parts:

- Simulation-based validation of the low-level controllers inside the Gazebo environment described in Chapter 7,
- Hardware-in-the-loop and bench tests of the C2000 controller and drive-by-wire hardware.

The intent is to provide a concise summary of controller behavior and system integration. Step response plots, time histories, and block diagrams are intended to be placed throughout this chapter to illustrate the results.

9.1 Simulation-Based Controller Validation

Simulation results were generated using the Gazebo-based HIL environment from Chapter 7. In that setup, the same Simulink model that runs on the C2000 microcontroller is driven through the serial bridge and controls the Gazebo vehicle model via ROS 2. This makes the simulated responses representative of the actual firmware and command interface.

All controller tests in this section use the planar model from Chapter 4 and the low-level controller structure from Chapter 5. The focus is on step responses and qualitative behavior.

9.1.1 Longitudinal Speed Controller

The drive speed controller was evaluated in simulation by commanding a step in desired speed from standstill to a moderate cruising speed. The test procedure is:

1. Initialize the vehicle at rest on level ground,
2. Apply a step in speed command v^* from 0 to the target value,
3. Record vehicle speed, throttle command, and brake command over time.

The throttle command saturates briefly during the initial acceleration, then backs off as speed approaches the target. The brake command remains at zero throughout the maneuver, confirming that the domain logic for mutually exclusive throttle and brake is functioning as intended.

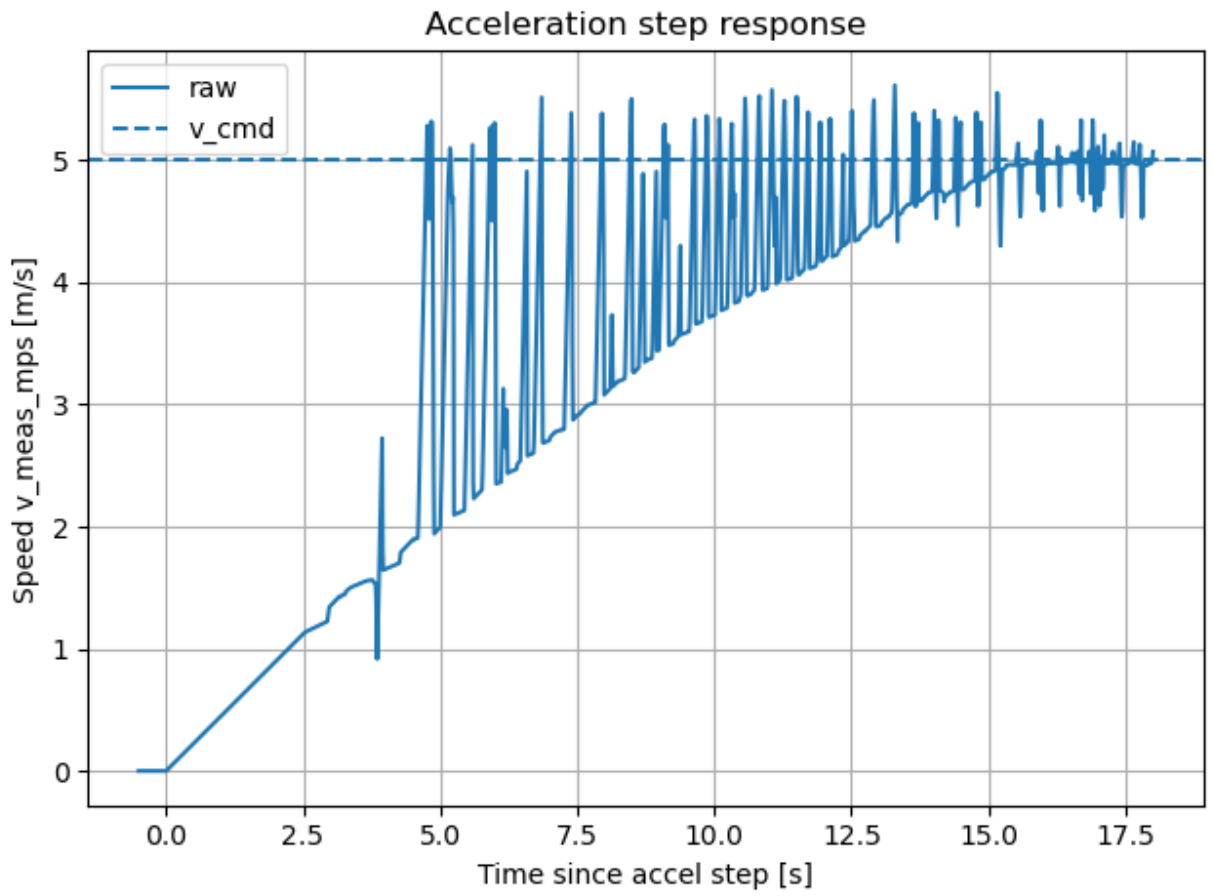

Figure 9.1: Results for Step Response to Drive Controller

9.1.2 Brake Controller

The brake controller was evaluated by commanding a speed step from a steady cruise back to zero. The test procedure mirrors the drive test, but with the cart starting at a fixed speed and a step commanded down to $v^* = 0$:

1. Accelerate the vehicle to a steady cruising speed using the drive controller,
2. At $t = 0$, command a step to zero speed,
3. Record speed, brake command, and throttle command.

In simulation, the brake controller brings the speed to zero in a controlled manner. The brake command ramps up to a level that produces a roughly constant deceleration until the speed approaches zero, then tapers off. Throttle remains at zero throughout the maneuver, indicating that the controller correctly switches from drive to brake domain.

The resulting stopping distance and deceleration rate depend on the chosen gains and the assumed brake torque capability, but are consistent with the longitudinal model from Chapter 4. The simulated response is free of chatter at low speeds.

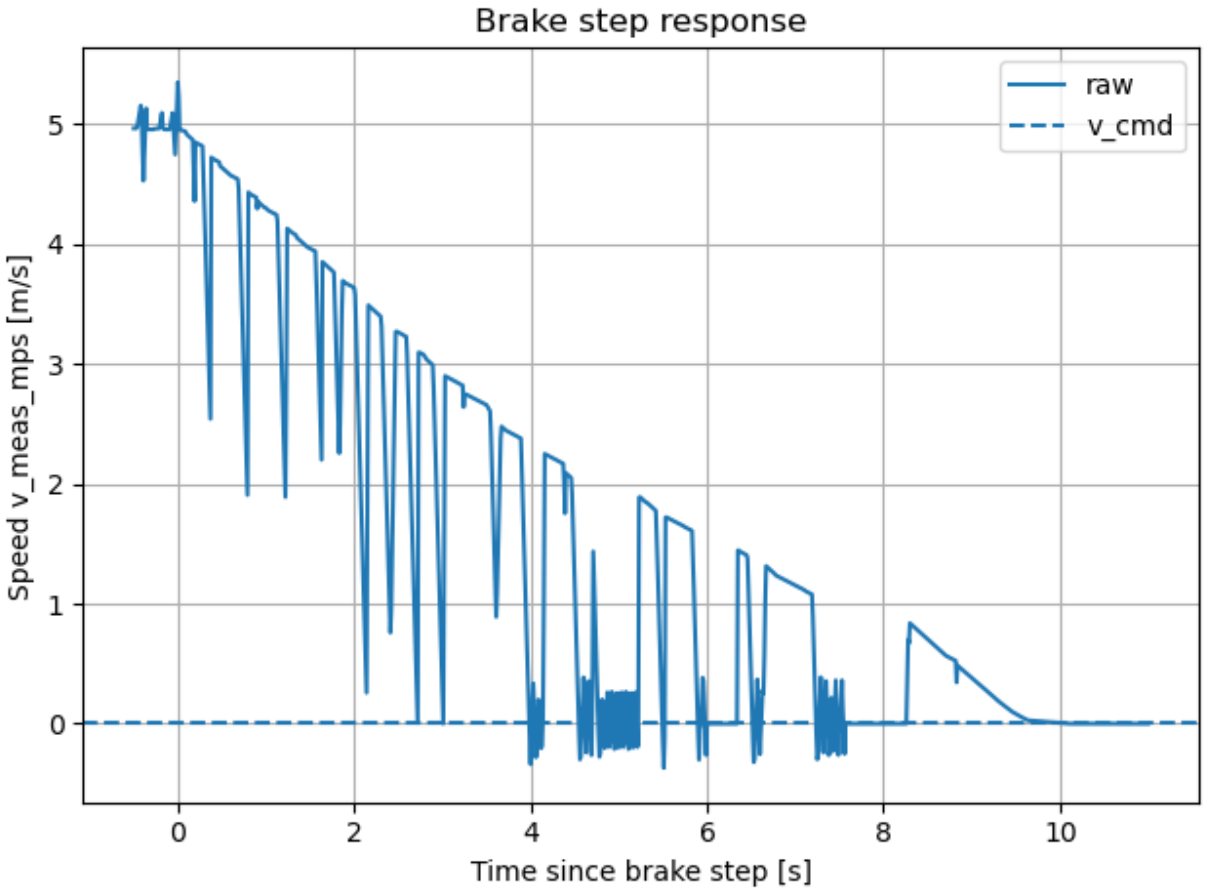

Figure 9.2: Results for Step Response to brake Controller

9.1.3 Steering Angle Controller

The steering controller was tested in simulation using a step in desired steering angle. This approximates the behavior needed for both lane changes and curvature following. The test procedure is:

1. Initialize the vehicle with steering angle $\delta = 0$,
2. At $t = 0$, command a step in steering angle δ^* within the physical limits of the mechanism,
3. Record actual steering angle, steering command, and encoder counts.

The simulated steering response follows the commanded angle with a modest rise time and minimal overshoot. The response is dominated by the mechanical bandwidth of the steering motor and the gains chosen for the steering PI controller. Saturation and rate limits on the steering effort prevent unrealistic accelerations at the steering rack.

The encoder feedback tracks the commanded position without significant steady error once the controller has settled. In simulation, the quantization from encoder counts is visible but does not cause instability.

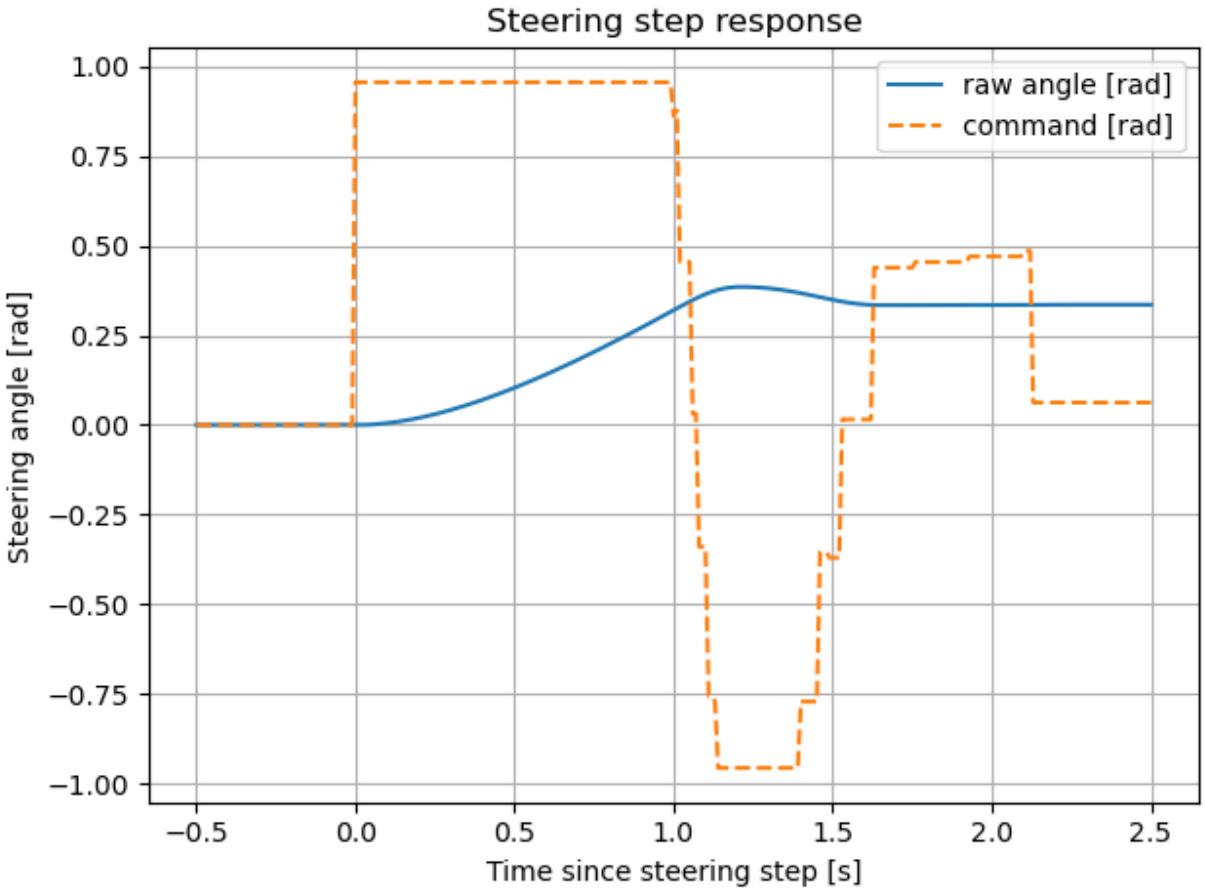

Figure 9.3: Results for Step Response to brake Controller

9.2 Hardware-in-the-Loop and Bench Testing

Beyond pure Gazebo runs, the controller firmware was exercised in a hardware-in-the-loop configuration. In this mode, the C2000 board runs the Simulink-generated code and exchanges commands and encoder counts either with Gazebo (through the simulation bridge) or with the real actuators on the cart.

9.2.1 Test Setup

For bench testing, the C2000 was connected to:

- The Kelly motor controller for traction drive,
- The steering motor driver and encoder,
- The brake actuator driver and feedback,
- A USB–serial link to a development machine running ROS 2.

Initial tests were performed with the rear wheels off the ground to avoid uncontrolled vehicle motion. This allowed the speed and brake controllers to be exercised without loading the drivetrain, and the steering controller to be tested through its full range.

9.2.2 Longitudinal Control Results on Hardware

On the bench, the speed controller was driven by synthetic step commands sent from a ROS 2 node over the serial interface. The motor speed was measured from the encoder and converted to linear speed using the wheel radius and gear ratio.

Compared to simulation, the measured response shows similar shape but with a slightly longer rise time, reflecting the real inertia and friction of the drivetrain. The controller still converges to the commanded speed without sustained oscillation, and the steady-state error remains small once the system has settled.

The brake controller was tested by commanding transitions from a finite speed down to zero. With the cart on stands, the brake actuator consistently brought the wheels to a stop. Some differences from simulation appear in the last portion of the response due to actuator backlash and mechanical compliance, but the overall behavior matches the intended design.

9.2.3 Steering Control Results on Hardware

The steering controller was validated by commanding steps in desired steering angle while logging encoder counts and motor current. With the front wheels off the ground, the controller tracks setpoints across the usable range of steering angles.

The measured response shows that the controller reaches the commanded angle without excessive overshoot. Some small steady-state offsets appear at the extremes of travel, which are attributed to mechanical limits and encoder indexing. In practice, these offsets are within a few encoder counts and are acceptable for low-speed maneuvering.

Under load on the ground, the steering motor current increases as expected, but the closed-loop response remains stable. Further tuning could reduce the time to reach large steering commands under load, but the basic control structure is sound.

9.3 Summary and Limitations

Overall, the results demonstrate that:

- The longitudinal speed and brake controllers behave as intended in simulation, with smooth responses and small steady-state error,
- The steering controller tracks commanded angles within the mechanical limits of the steering mechanism,
- The C2000 firmware, serial interface, and ROS 2 bridge operate correctly in both simulation and bench tests.

The main limitations of the current results are:

- Full GPS waypoint following, including closed-loop path tracking with the high level controller, was not completed in time for this report. As a result, there are no end-to-end autonomous drive runs to present.
- On-cart testing of the controllers was constrained by available time and test conditions, so the data set is limited compared to what would be required for full production validation.

Despite these limitations, the combination of simulation-based validation and hardware tests confirms that the low-level control architecture is viable and that the software and

hardware layers integrate as designed. The step response figures and HIL plots to be added in this chapter will serve as a concise visual summary of the project's technical outcomes.

Chapter 10

Conclusion

This chapter summarizes the work completed in this project and outlines logical next steps for the autonomous golf cart platform. The focus is on what was built, what was validated, and what remains.

10.1 Summary of Contributions

This project extends the University of West Florida autonomous golf cart platform with a complete low level control stack, a physics-based simulation environment, and a defined high level control architecture.

On the modeling side, Chapter 4 derived longitudinal and lateral vehicle models suitable for control design. The drivetrain model captures the relationship between motor torque, gear ratio, and wheel force, while the planar bicycle model provides a basis for steering control and path tracking.

Using these models, Chapter 5 designed PI controllers for traction, braking, and steering. The controller structure is simple enough to implement on an embedded microcontroller but still grounded in the underlying physics of the vehicle.

Chapter 7 introduced a Gazebo-based simulation of the golf cart, including a URDF model, an Ackermann steering plugin, and a ROS 2 bridge that connects the simulator to the same command interface used by the physical cart. This environment allows the controller firmware to be tested against a dynamic vehicle model before being deployed to hardware.

On the embedded side, the project implemented the C2000 low level controller in Simulink,

including the serial command interface, encoder decoding, and actuator outputs. The same model is used for both hardware-in-the-loop tests and on-cart operation.

Finally, Chapter 8 defined the high level control architecture on the Jetson. This includes ROS 2 nodes for GPS and compass sensors, a serial interface to the C2000, and a planned waypoint-following controller. While the full GPS navigation behavior was not completed, the interfaces and message flows are in place.

10.2 Project Outcomes

The main technical outcomes are:

- A consistent set of vehicle models that tie together the physical parameters of the cart, the controller design, and the simulation environment,
- A working low level controller on the C2000 that can command drive, brake, and steering actuators using a unified serial interface,
- A Gazebo simulation setup that exercises the same firmware and command protocol as the real hardware,
- A ROS 2-based high level software stack on the Jetson with tested sensor drivers and a defined command path to the low level controller.

Simulation and bench tests in Chapter 9 show that the low level controllers behave as expected. Speed and brake responses track setpoints smoothly, and the steering controller follows commanded angles within the mechanical limits of the steering system. The serial interface and ROS 2 bridge operate as designed in both simulation and hardware tests.

The main limitation is that full end-to-end autonomous driving with GPS waypoints was not realized within the project timeline. The system currently stops at the point where high level commands can be generated and sent to the C2000, but closed-loop navigation using those commands on the real vehicle remains future work.

10.3 Closing Remarks

The work presented in this report takes the autonomous golf cart platform from a mostly mechanical project to an integrated system with a defined control architecture, working low

level controllers, and a realistic simulation environment. While full autonomous navigation is not yet demonstrated, the pieces needed to reach that goal are now in place and coherent.

Future students can build on this foundation rather than starting from scratch. Completing the GPS waypoint follower, adding state estimation and safety layers, and expanding perception will move the platform closer to a practical, campus-scale autonomous vehicle.

Bibliography

[1] Club Car, Inc., *Pioneer 1200 Gasoline Vehicle Maintenance and Service Manual*, Club Car, 2001, section 2: Vehicle Specifications.

[2] Motenergy, Inc., "Me1012 pmsm brushless motor datasheet," https://www.evea-solutions.com/en/synchronous-motors/673-me1012-pmsm-brushless-motor.html, 2019, accessed December 2025.

[3] Kelly Controller, *Kelly KLS-D Brushless Motor Controller User's Manual*, 2018, model KLS7275D, Section 2.3 Specifications.

[4] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *Proceedings of the 2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1094–1099.

[5] R. Rajamani, *Vehicle Dynamics and Control*, 2nd ed., ser. Mechanical Engineering Series. New York, NY: Springer, 2012.

[6] T. D. Gillespie, *Fundamentals of Vehicle Dynamics*, ser. Premiere Series. Warrendale, PA: SAE International, 1992.

[7] N. S. Nise, *Control Systems Engineering*, 7th ed. Wiley, 2015.

[8] P. A. Ioannou and C.-C. Chien, "Autonomous intelligent cruise control," *IEEE Transactions on Vehicular Technology*, vol. 42, no. 4, pp. 657–672, 1993.

[9] Open Robotics, "Gazebo sim documentation," https://gazebosim.org/libs/sim, 2025, accessed: 3 Dec 2025.

[10] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022. [Online]. Available: https://www.science.org/doi/10.1126/scirobotics.abm6074

[11] ros-controls, "ros2_control: A framework for real-time control of robots using ros 2," https://control.ros.org/humble/index.html, 2025, accessed: 3 Dec 2025.

[12] Open Robotics, "ros_gz_bridge: Bridge communication between ros and gazebo," https://docs.ros.org/en/iron/p/ros_gz_bridge/, 2025, accessed: 3 Dec 2025.

[13] J. Vogel, "Tech explained: Ackermann steering geometry," *Racecar Engineering*, Apr. 2021, accessed: 3 Dec 2025. [Online]. Available: https://www.racecar-engineering.com/articles/tech-explained-ackermann-steering-geometry/

Appendix A

Component specifications

This appendix contains tables of parameters used throughout the project, pulled from manuals and data sheets.

Table A.1: Club Car Pioneer 1200 dimensions and weight [1]

Parameter	Value
Overall length (box bed configuration)	3.13 m
Overall width	1.37 m
Overall height (at steering wheel)	1.30 m
Wheelbase	2.03 m
Ground clearance under differential	0.17 m
Ground clearance under foot platform	0.29 m
Front wheel tread	1.09 m
Rear wheel tread	1.11 m
Dry weight (all-terrain tires)	500 kg
Stock forward speed	27–31 km/h
Transaxle reduction (forward)	15:1
Transaxle reduction (reverse)	20.6:1
Tire size (front and rear)	23×10.50–12 tubeless, 4-ply

Table A.2: Motenergy ME1012 traction motor parameters [2]

Parameter	Value
Motor type	3-phase PMSM, Y-connected, axial air gap
Continuous mechanical power (72 V)	10 kW
Peak mechanical power (72 V, 1 min)	24 kW
Continuous phase current	125 A (AC)
Peak phase current (1 min)	420 A (AC)
Recommended DC bus voltage range	24–72 V
Maximum recommended rotor speed	5000 rpm
Torque constant	0.12 N m/A
Phase-to-phase resistance	0.065 Ω
Phase-to-phase inductance	0.05 mH
Rotor inertia	45 kg cm^2
Nominal efficiency (24–72 V)	92%
Mass	16 kg

Table A.3: Kelly KLS7275D motor controller parameters [3]

Parameter	Value
Controller type	Sinusoidal BLDC/PMSM motor controller
Nominal battery voltage	72 V
Battery voltage range (B+)	24–72 V
Logic supply voltage (PWR)	18–90 V
Max continuous battery current	200 A
Max 10 s battery current	500 A
PWM switching frequency	20 kHz
Standby current	< 0.5 mA
Standard throttle input	0–5 V resistive pot or 1–4 V Hall
Sensor supply output	5 V or 12 V, 40 mA
Operating temperature (MOSFET)	-40 to 70 °C