****

**EE6008 Collaborative Research and Development Project**

**Project Report**

**<< A balancing mobile robot>>**

**Students’ Names: <<Ju Jialin, Huang Xiaoying, Huang Jian, Hou Yabin, Liu Zhaoyue>>**

**Supervisor’s Name: <<Ling Keck Voon>>**

**School of Electrical and Electronic Engineering**

**Academic Year 2024/25**

**Semester 1**

# Table of Contents

1. Purpose / Project Objectives
2. Project Summary
3. Scope
4. Schedule
5. Cost / Benefit
6. Outcome / Expected Outcomes
7. Individual Reports from Team Members
8. References

Appendix - Project Members Information

1. **Purpose/ Project Objectives**

In recent years, advancements in artificial intelligence and big data have accelerated the integration of smart technologies into everyday life. Robots are increasingly becoming a part of human life, with applications ranging from robotic vacuum cleaners and logistics robots to hospitality robots and educational assistants. Compared to legged robots, wheeled robots represent a more mature and reliable mode of robotic mobility, especially for applications requiring speed and stability.

The goal of this project is to develop a two-wheeled balancing mobile robot as a practical demonstration of control system design and its embedded implementation. Through this hands-on experience, students will have the opportunity to apply advanced control methods learned in their MSc studies in Computer Control and Automation (CCA) and related courses, fostering a deeper understanding of control systems in real-world applications.

The two-wheeled balancing robot, with its dual-wheel coaxial design and independent drive system, offers high maneuverability and adaptability across various terrains. It is capable of in-place rotation and adjustable-radius turning, making it more flexible and responsive compared to traditional multi-wheeled systems. Additionally, with a compact footprint, the robot can operate effectively in confined spaces. Its simple structure and low power requirements make it an accessible tool for short-distance transportation and personal mobility.

This project allows students to innovate beyond the basic design of the balancing robot, encouraging creativity and exploration of enhanced functionality. Through building and extending the reference model, students can gain practical insights into the potential of control systems in modern robotics, supporting the broader vision of integrating intelligent robotic solutions into everyday life.

1. **Project Summary**

This robot is designed to achieve autonomous balance by processing data from a gyroscope and accelerometer to detect the robot’s posture and applying a PID control algorithm in real-time to maintain stability. With remote control, the robot can smoothly move forward, backward, and turn. Additionally, it can follow a path and avoid obstacles based on data from an ultrasonic sensor. The project is structured into the following sections:

* Mechanical Design of the Balancing Robot: This section focuses on the mechanical structure, center-of-gravity adjustments, and electrical design aspects necessary to ensure stability and responsiveness in the robot's movement.
* Signal Processing: Key tasks include data acquisition and processing from the MPU6050 (gyroscope and accelerometer) for posture detection, HC-SR04 (ultrasonic sensor) for obstacle detection, and motor encoder for precise movement tracking.
* PID Control Algorithm: Using the processed data, the PID control algorithm calculates the upright position, speed, and direction control loops, enabling stable and responsive movement.
* Remote Control Transmission: The Bluetooth module enables remote data reception from a mobile device or remote controller, allowing external control over the robot's movements.
* Display Interface: The display screen provides real-time feedback on the robot's status, including posture, speed, and any detected obstacles, enhancing both user interaction and debugging processes.

Each section collectively contributes to the robot’s autonomous balancing, navigation, and user-controlled functionalities, demonstrating the integration of mechanical, electronic, and software components in achieving the project objectives.

1. **Scope**
2. **Hardware Design**

**1.1 Overall Hardware System Design**

This project's hardware setup is illustrated in the diagram, including the ESP32 controller, MPU6050 sensor, motor driver module, motors, ultrasonic module, power supply, and Bluetooth module. At the core of the system is the ESP32, which gathers real-time orientation data from the MPU6050 via I2C to determine the self-balancing car's balance and position. Based on this data, the ESP32 sends precise PWM and directional signals to the motor driver, controlling motor speed and direction to maintain stability.The ultrasonic module, built with the HC-SR04 sensor, continuously monitors obstacles in front of the car, transmitting distance data to the ESP32. Based on this distance information, the ESP32 decides whether to slow down, steer, or stop to avoid collisions. Additionally, the Bluetooth module enables wireless communication with a mobile device, allowing for remote control and real-time adjustments. Finally, the power supply module ensures that all components receive a stable power source, supporting consistent system operation.

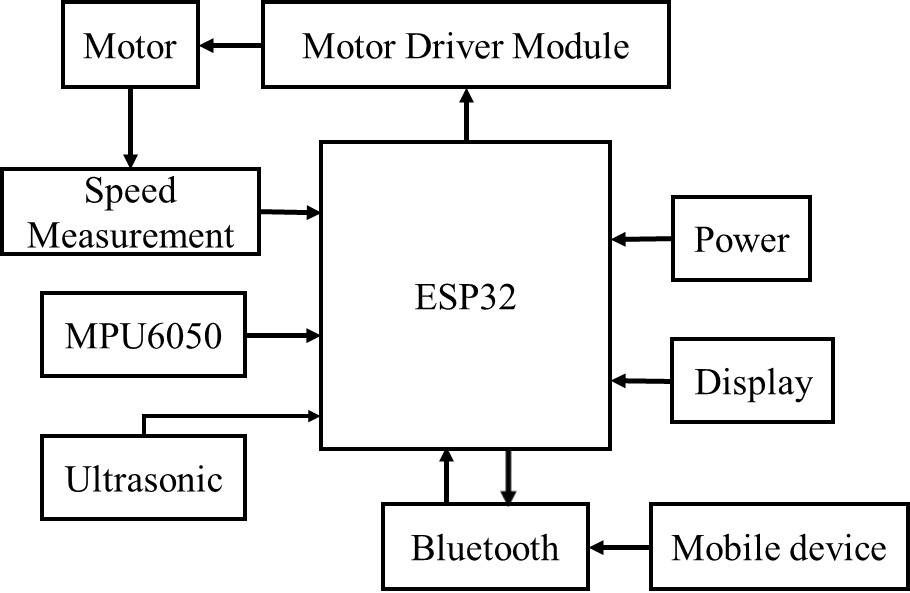


Fig. 1.1 Overall System Hardware Scheme

**1.2 Hardware Modules Overview**

ESP32 Minimal System Board: The ESP32S3 module functions as the main control unit. The EN pin is connected to a power-on reset circuit, which prevents instability caused by fluctuating power levels and ensures reliable operation of the module. The TXD0 & RXD0 interface manages communication with external devices, while various GPIO pins connect to other modules for extended functionality.

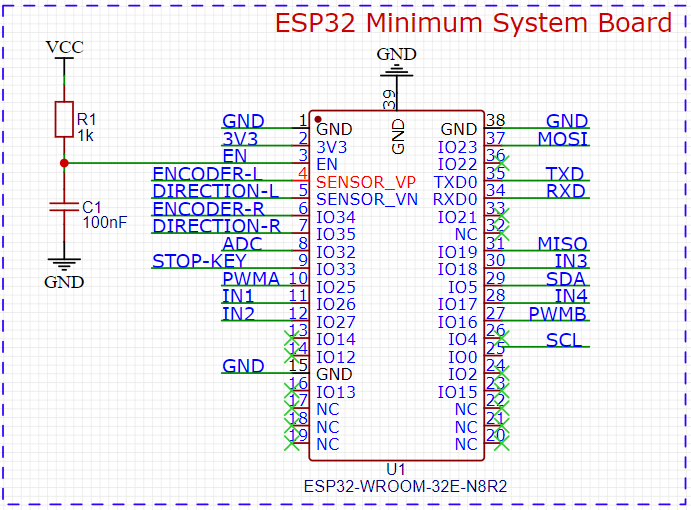


Fig. 1.2 ESP32 Minimum System Board

This module is responsible for orientation detection, integrating a 3-axis accelerometer and gyroscope. It communicates with the ESP32 via the I2C interface, providing real-time angle and acceleration data essential for maintaining balance. To stabilize the power supply, a decoupling capacitor (C5) is connected between VDD and ground, effectively reducing noise interference. Both I2C lines (SDA and SCL) are connected to pull-up resistors to maintain a high signal level on the bus. Additionally, a small capacitor is connected to CPOUT to filter out high-frequency noise on the I2C clock line, ensuring stable data transmission.

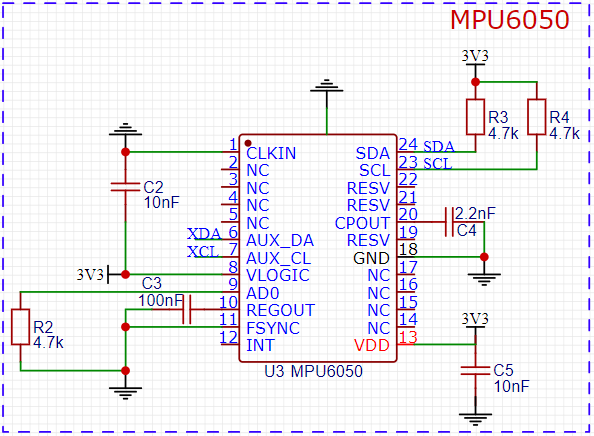


Fig. 1.3 MPU6050 Sensor Module

Motor Driver Module: Using the TB6612FNG driver, this module receives PWM signals to adjust motor speed and uses direction pins for forward/reverse control. It’s essential for stable motor control, helping the car maintain balance.

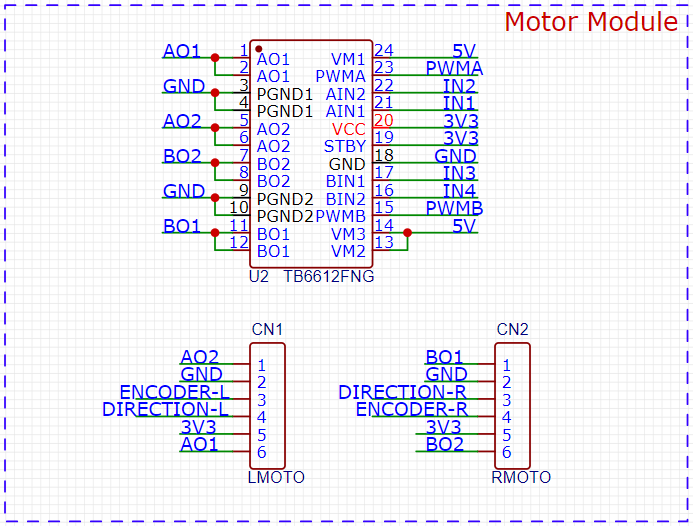


Fig. 1.4 Motor Drive & Connector Module

Motor Connector Module: Connectors are used to link the motors with the driver module. Separate connectors for the left and right motors make assembly and maintenance easier.

Ultrasonic Wave Module: Using HC-SR04 to measure distance by sending ultrasonic pulses from the TRIG pin and receiving echoes on the ECHO pin. Powered by 5V (VCC) with GND connected, it calculates distance based on the time between pulse emission and echo return, making it ideal for obstacle detection.

1. **Kinetic Model**

The self-balancing robot maintains stability by controlling motor-driven wheels to adjust angular velocity in real time, counteracting deviations in its center of gravity. It is commonly modeled as an inverted pendulum, a classical unstable system with the center of gravity above the pivot point, requiring external force to return to balance.

Similar to the inverted pendulum, the robot tilts or deviates when disturbed. Using this model helps analyze the robot’s motion and control needs. Achieving self-balance depends on ensuring that wheel acceleration is linearly related to the tilt angle, generating a restoring torque to counteract deviations, which forms the basis for balance controller design and improves stability.

**2.1** **Motion Assumptions and Model Simplifications**

2.1.1 Motion Assumptions

* Linear Motion: The robot moves along the y-axis in the y-o-z plane, with steering disregarded. This simplifies the model to focus on forward-backward stability control, and the schematic diagram is shown in Fig.2.1。

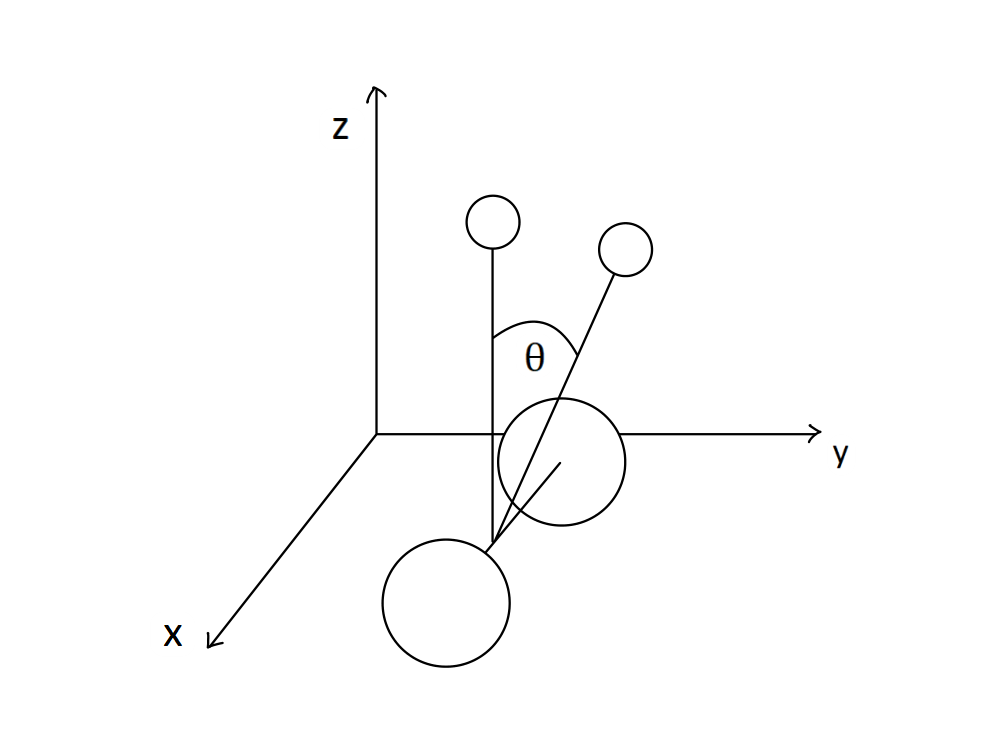


Fig. 2.1 Motion Assumptions schematic diagram

* External Interference: Assumes external disturbances are perpendicular to the robot’s rotational radius, ignoring lateral disturbances for simplicity.
* Initial State: The robot starts at rest with zero wheel acceleration and body tilt, providing a baseline for control system analysis.

2.1.2 Model Simplifications

The inverted pendulum is a classic unstable system, where the center of mass is located above the pivot point. Any deviation from the vertical position generates a restoring force due to gravity, attempting to bring the pendulum back to equilibrium. However, since the restoring force is aligned with the displacement, the system cannot return to balance on its own and requires external input for stabilization. Similarly, the self-balancing robot has its center of mass above the wheels, causing instability when tilted. Both systems share the need for external control input to maintain stability, the comparison chart is shown in Fig.2.2 and in the case of the robot, adjustments to wheel speed are required in real-time to correct the tilt angle and keep the robot balanced.

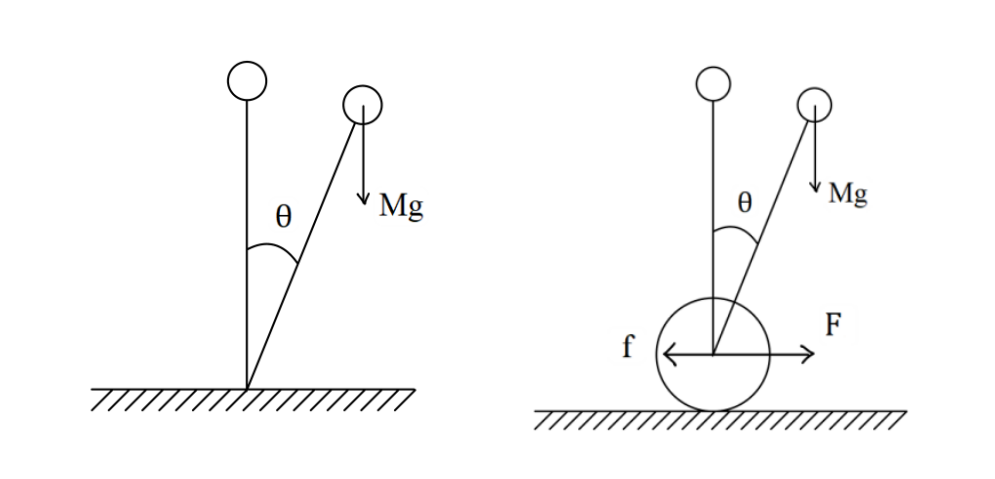


Fig. 2.2 Inverted Pendulum and Balancing Mobile Robot Force Comparison Digram

The inverted pendulum model simplifies the dynamics of the self-balancing robot by describing the system with a second-order differential equation based on the pendulum's angle and angular velocity. In the robot, a similar model can be derived by considering the tilt angle of the body and the wheel acceleration. This model helps determine how to apply control inputs, such as wheel acceleration or angular velocity to generate enough restoring force to maintain balance. By linearizing the inverted pendulum model and applying control theories like PID, an effective balance controller can be designed to keep the robot stable.

**2.3 Force Analysis of the Self-Balancing Robot**

2.3.1 Force Analysis of the Wheels

To analyze the forces acting on the wheels, we decompose the wheel’s movement into translational and rotational components.

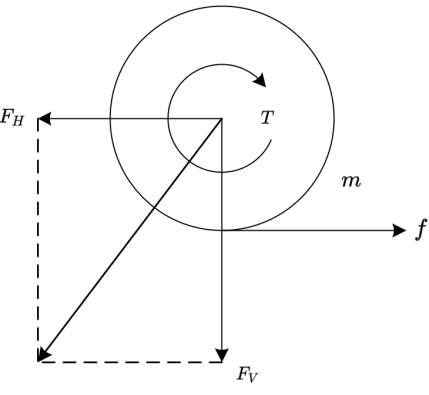


Fig. 2.3 Force Analysis of the Wheels

According to Newton's second law, the horizontal translational force on the wheel can be expressed as:

Using the rotational dynamics equation for rigid bodies, we obtain:

The relationship between translational and rotational motion of the wheels can be expressed as:

Combining the translational and rotational dynamics of the wheels, we derive:

Given that the mass of the wheel is significantly smaller than the mass of the body and has minimal impact on the robot’s motion, the mass m of the wheel can be ignored, resulting in:

Thus, the relationship between the driving force and the horizontal force acting on the self-balancing robot conforms to the standard inverted pendulum model.

2.3.2 Force Analysis of the Body

For the robot body, forces can be divided into translational and rotational components as well.

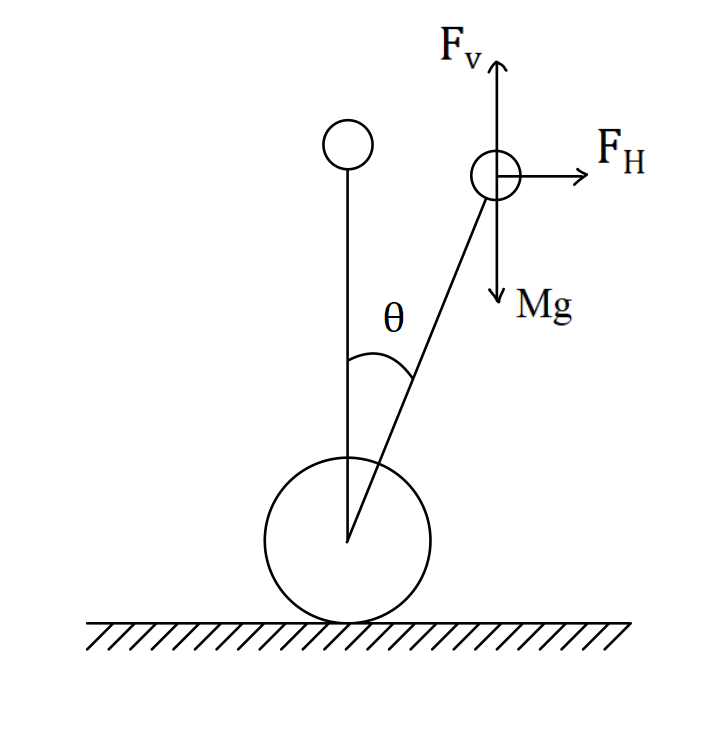


Fig. 2.4 Force Analysis of the Body

Analyze the forces acting in the horizontal and vertical directions of the body to determine the resulting translational dynamics.

Based on the rotation of the body, the rotational dynamics can be described as:

Combining translational and rotational dynamics, we obtain the complete equations of motion for the system.

Symbol Definition Table is below

Table 2.1 Symbol Definition Table for System Dynamics

|  |  |
| --- | --- |
| Symbol | Definition |
| *m* | Mass of the wheel |
| *M* | Mass of the body |
| *g* | Gravitational acceleration |
| *T* | Motor torque |
| *x* | The distance the wheel moves |
| *ω* | Angular velocity of the wheels |
| *f* | Frictional force |
|  | The horizontal component of the force |
|  | The vertical component of the force |
|  | Driving force |
| *r* | Wheel radius |
|  | Angle of inclination |
|  | Height of center of mass |

**2.4 Dynamic Mathematical Model Formulation**

Linearized around the equilibrium point, can get:

System Equations:

Applying the Laplace transform to the equation, we get:

The transfer function with driving force as input and car inclination Angle as output is

Defining the state variables:

we derive the system's state-space equations:

**2.5 System Analysis**

Parameter Measurement and State-Space Equation Calculation

To accurately model the dynamics of the self-balancing car system, it is essential to precisely measure key physical parameters, such as the mass of the vehicle body and the height of the center of mass. These parameters directly influence the system's dynamic and control characteristics.

After measurement, we obtain values *M=1.44kg*, *l=3cm*, which are then substituted into the system's dynamic equations to derive specific numerical values for the matrices A, B, C and D in the state-space model.

Similarly, we can also get the system transfer function as:

2.5.1 Stability Analysis of the System

The self-balancing car can be modeled as a classic inverted pendulum, which exhibits the characteristics of an unstable system. In the state-space representation of this system, the eigenvalues of matrix A determine its stability.

It is calculated that:.

The eigenvalues are not all in the left half plane, the system is unstable.

Through the transfer function, the open loop impulse response can be obtained.

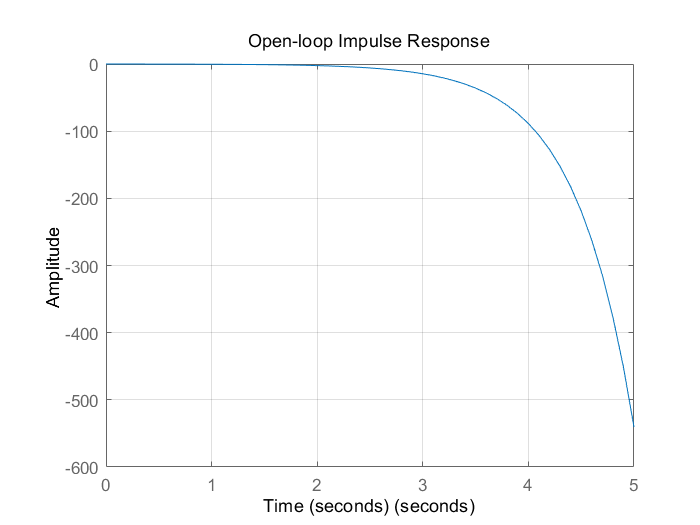


Fig. 2.5 Open Loop Impulse Response of the System

Without control input, the system will diverge from equilibrium and cause the vehicle to fall. To ensure stability, a controller must be designed so that the eigenvalues of the closed-loop system matrix lie in the left half of the complex plane, ensuring asymptotic stability. By calculating the eigenvalues of the state matrix A, we can assess whether the system is stable in its natural state and can maintain equilibrium under control.

2.5.2 Controllability and Observability Analysis of the System

Controllability and observability are essential metrics for designing an effective control system for the self-balancing car. Controllability determines whether the system can drive the vehicle to any desired state through input control signals. In the state-space model, the controllability matrix is constructed from matrices A and B, and its rank reveals whether the system is fully controllable. A fully controllable system allows the self-balancing car to reach any position within the state space with appropriate control inputs. For an inverted pendulum system, full controllability is crucial for real-time adjustment of both the tilt angle and position to maintain stability.

Observability, on the other hand, assesses whether the system's internal states can be inferred from its output signals. The observability matrix, generated by matrices A and C, indicates the system’s observability through its rank. If the rank matches the state dimension, the system is fully observable, meaning all internal states can be accurately deduced from output measurements. In the context of the self-balancing car, full observability ensures real-time feedback on critical state information, such as tilt angle and angular velocity, enabling the controller to maintain dynamic balance and achieve stable control.

1. **Control Algorithm Design**

**3.1 Purpose**

3.1.1 Verify the Control Policy and Algorithm

Simulink simulation allows us to validate and optimize control strategies and algorithms before they are implemented in actual hardware. This simulation ensures that the control system functions as expected and that potential problems can be identified and corrected at an early stage of design.

3.1.2 Reduce Design Cost and Time

In the design of control system, it is often cumbersome to debug directly on the physical hardware, and the adjustment of parameters is not convenient, so it takes a long time. Simulink simulation can first verify the feasibility of the design, avoid the waste of hardware resources, and reduce the development time and cost. Simulation can quickly iterate in a virtual environment, optimize designs, and troubleshoot errors.

More importantly, reasonable initial pid parameters can be obtained through Simulink simulation, and the hardware parameters are fine-tuned on this basis, thus greatly reducing the trial and error cost of hardware tuning.

3.1.3 More Intuitive Analysis of System Dynamic Response and Performance

Simulation can help designers analyze the dynamic response characteristics of the system, such as steady state error, overshoot, rise time and response time, etc., and then adjust the controller to achieve the desired performance indicators. The control parameters can be easily adjusted and the performance of the system can be observed by simulation.

In addition, the existing tuning theory is often based on the system output waveform, which is not easy to measure in the hardware debugging process, and Simulink simulation can easily obtain the system output waveform.

**3.2 PID Parameter Design Method**

Due to the tedious process of hardware tuning, this project uses a combination of Simulink simulation and hardware tuning: First, the system is modeled, after obtaining the Simulink simulation model, the pid parameters are preliminarily estimated and the approximate range is determined, and then pid parameters are fine-tuned in the hardware.

3.2.1 Simulation Part

* Ziegler-Nichols method

Ziegler-Nichols method is a classical empirical parameter tuning method, which has two main types: step response method and critical proportional gain method.

* Step response method: applies to the step response curve of a known system.

Step1: Two parameters are determined by the step response curve of the system: time constant and delay time .

Step2: Select the corresponding PID parameters according to the ratio of and .

* Critical proportional gain method: suitable for closed-loop systems.

Step1: Gradually increase the proportional gain until the system has just sustained oscillation (i.e. critical oscillation). At this time, the proportional gain is called the critical proportional gain and the oscillation period is .

Step2: Obtain the preliminary PID parameters according to the empirical formula:

P controller:

PI controller:

PID controller:

This project uses the critical proportional gain algorithm.

3.2.2 Hardware

Since the establishment of the simulation model does not consider friction and other interference factors, the derivation of the dynamic model is also based on ideal conditions, and there are measurement errors in the physical parameters (such as the mass of the car , the center of gravity , etc.), there are not a small difference between the simulation model and the hardware. Therefore, after the initial parameters are obtained through simulation, the pid parameters need to be fine-tuned to get the ideal control effect. In this step, the parameters can be adjusted by observing the movement behavior of the car.

The following takes the vertical loop (PD controller) as an example to give the tuning process.

Step1: Adjust the proportional gain . The value ofaffects the response speed of the system, the larger the , the faster the response, and the smaller the , the slower the response of the car. over the size of the car will respond excessively, there will be a backward and forward oscillation phenomenon. In hardware debugging, in order to ensure safety, the initial parameter can be set small, and then gently push the car to tilt the car, observe the response speed of the car, if the car response has obvious lag phenomenon or no response before falling, the should be increased. So, adjust until the car appears shaking phenomenon.

Step2: Adjust the proportional gain . can suppress the vibration of the car, but if the value is too high, the car will be too sensitive to tilt and produce a high frequency of vibration. adjustment is similar to , using a smaller initial value and gradually increasing it. In this step, observe the oscillations of the car. As increases, high-frequency oscillations occur, and the power should be cut and reduced.

In addition, for, this parameter is mainly used in the velocity loop (or position loop) to eliminate steady-state errors. If is too small, there will be a steady state error to a certain extent, which is manifested as uniform linear motion (or deviation from the target position). If it's too big, the car will move backwards and forwards. Adjustments can be made according to this phenomenon.

**3.3 The Establishment of Simulation Model**

The control block diagram is as follows:

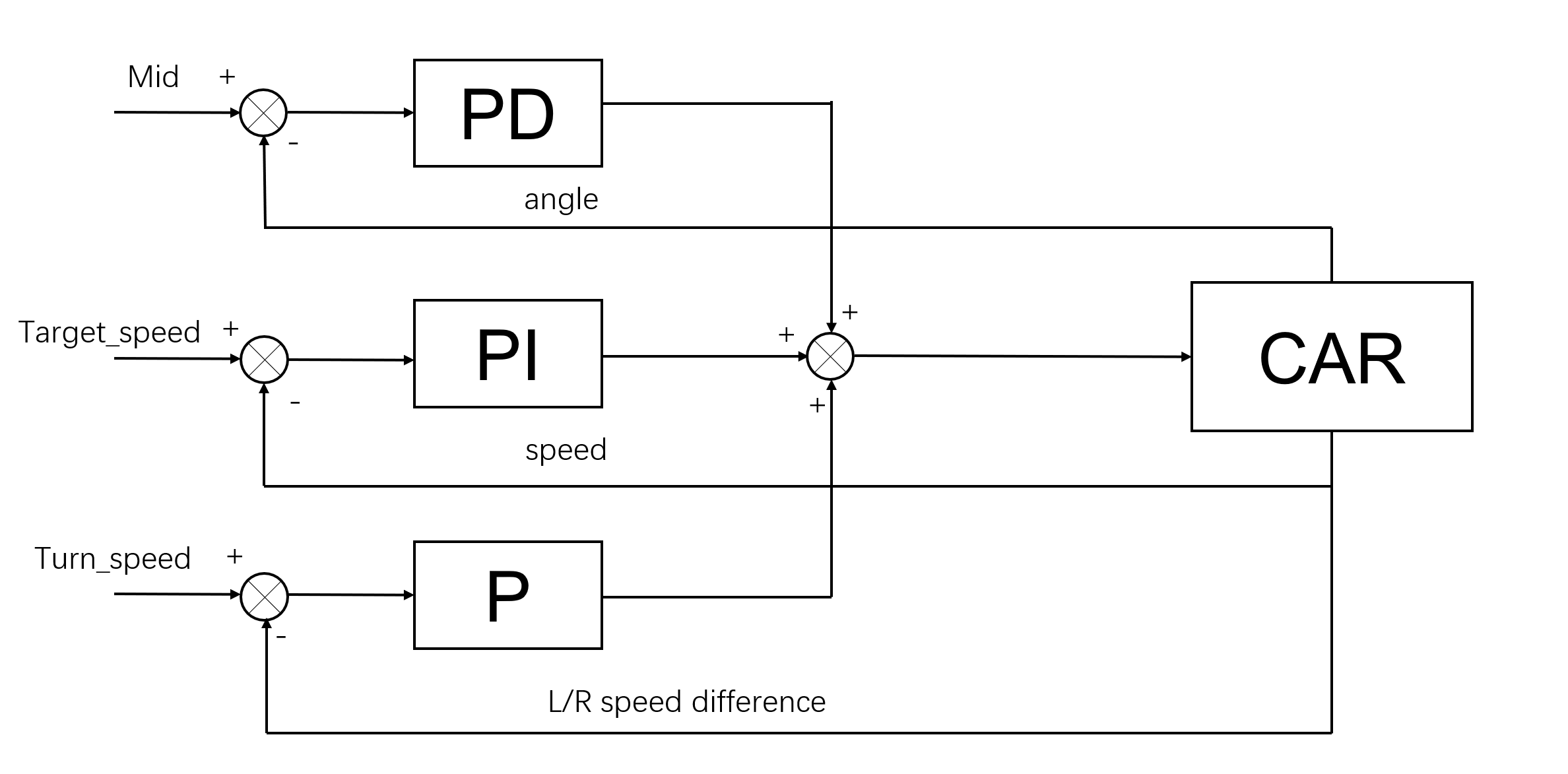


Fig. 3.1 PID Control Structure for Self-Balancing Robot

**3.2 Simulation Model**

The approximate state space model of the car is as follows:

Now that we have a state-space model of the system, we can model it using Simulink:

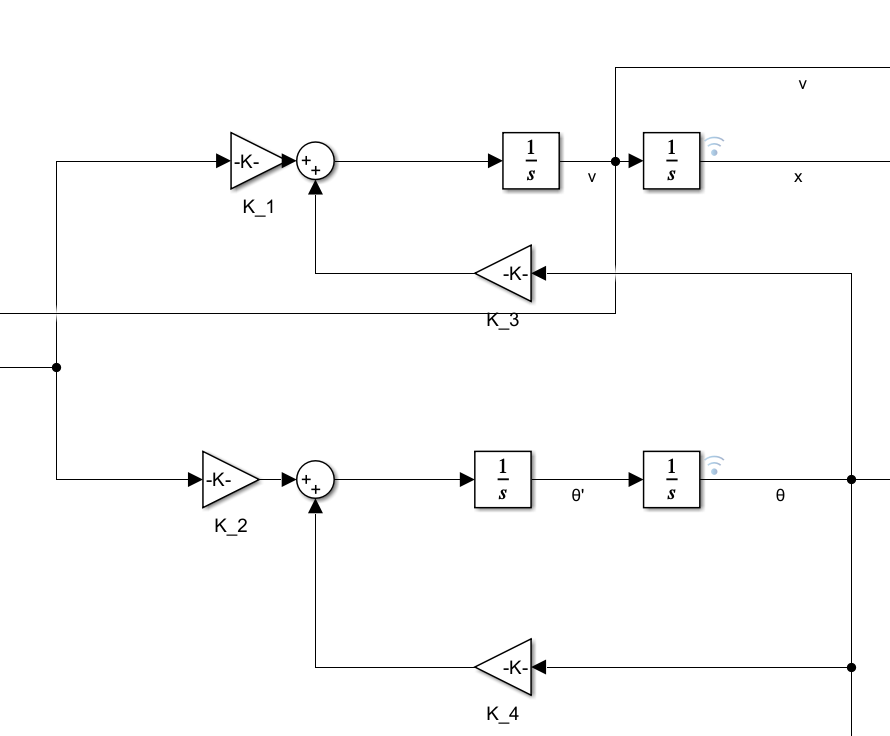


Fig. 3.2 Simulink Model of the State-Space Representation for Self-Balancing Robot

Where , , ,

Add the vertical PD controller:

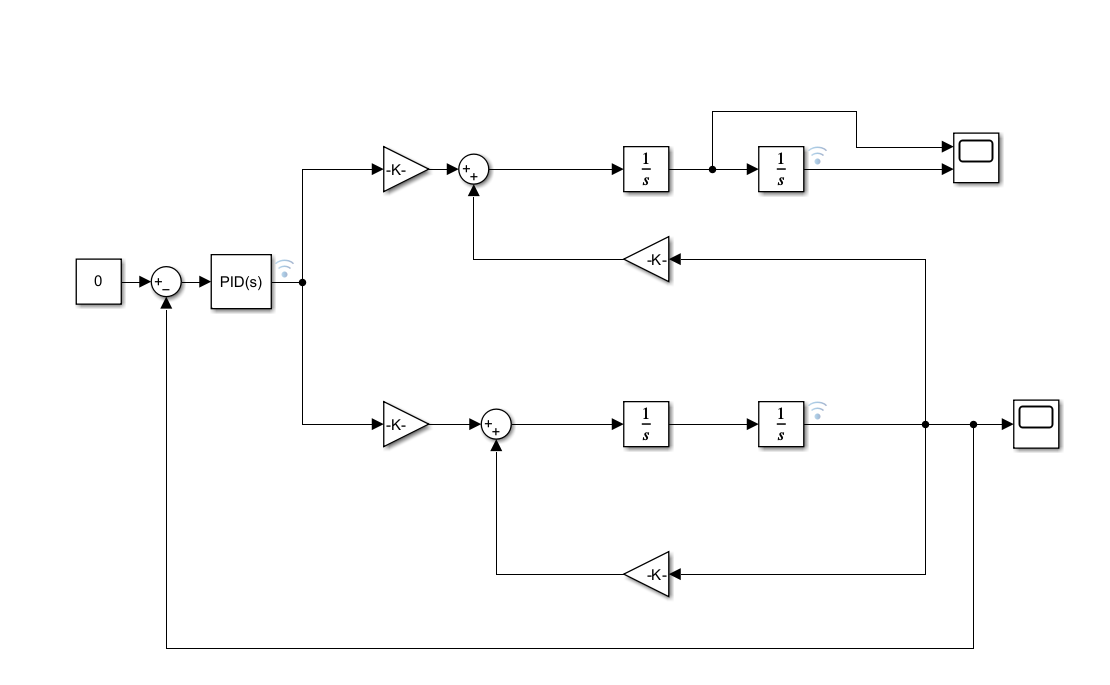


Fig. 3.3 Simulink Model of PID Control with Vertical PD Loop for Self-Balancing Robot

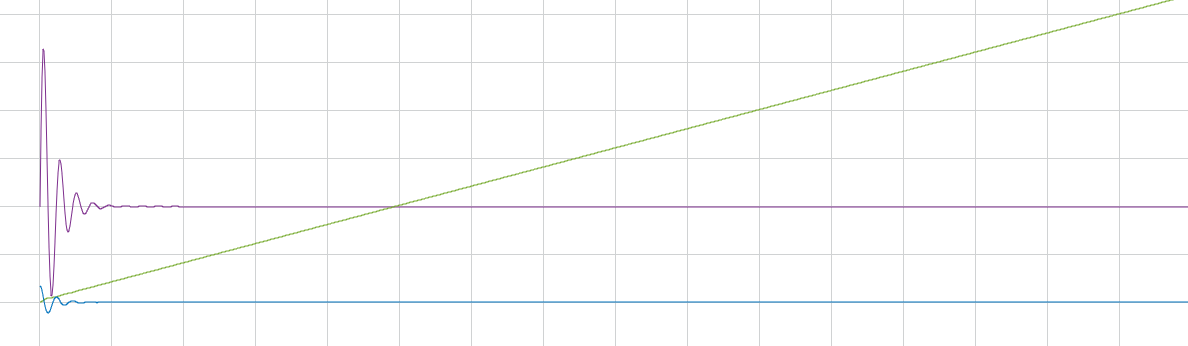


Fig. 3.4 Simulated Waveform of Angle-Blue, Velocity-Purple, and Position-Green

Add the speed PI controller

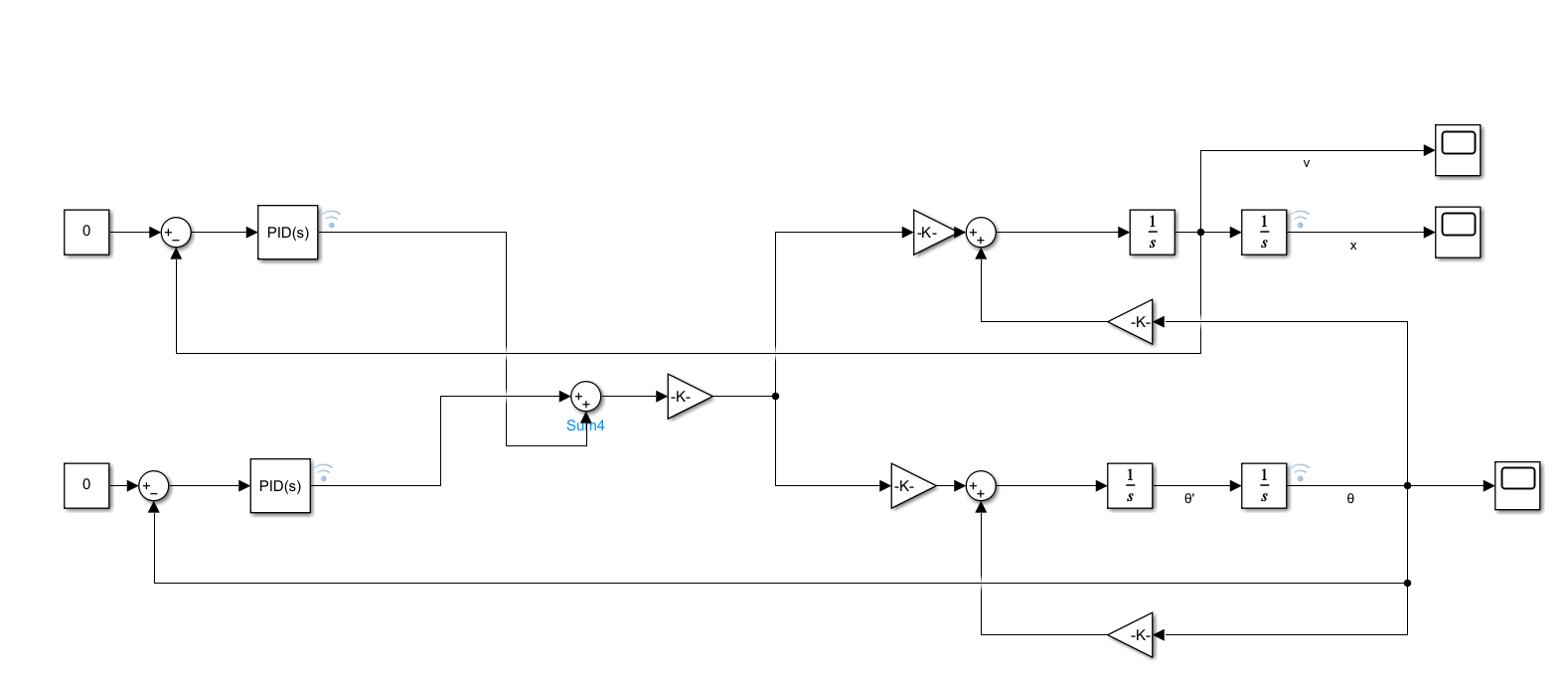


Fig. 3.5 Simulink Model of Vertical Loop and Velocity Loop with PI Controller

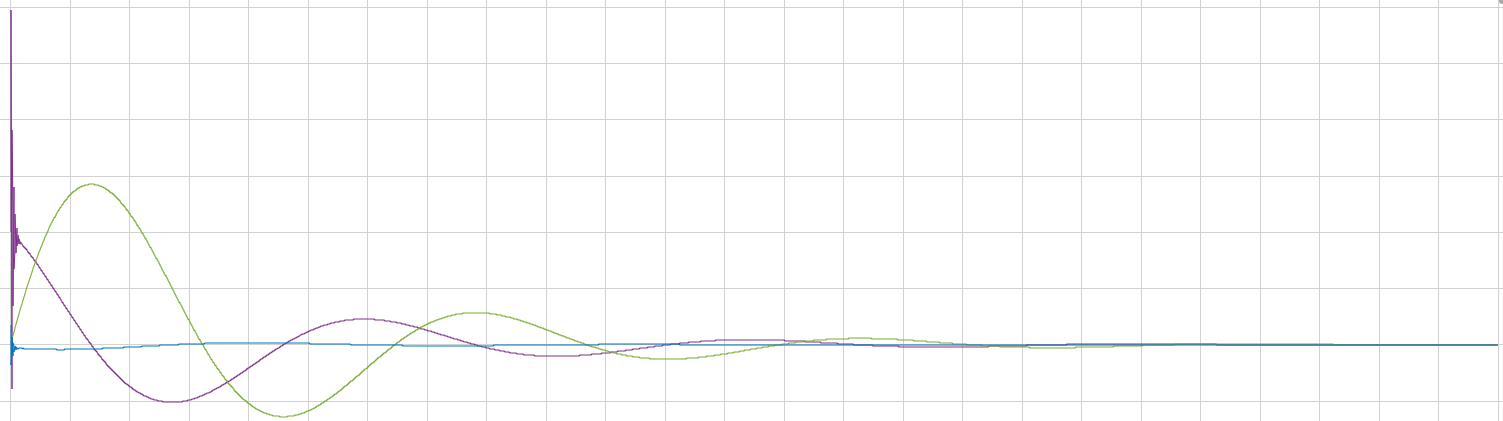


Fig. 3.6 Simulated Waveform for Angle-Blue, Velocity-Purple, and Position-Green with Speed PI Control

**3.3 Simscape 3D Modeling**

* Set the size and weight of the wheels and axle, and use the Rigid Transform module to achieve a 3-D rigid connection, then encapsulate it as a "wheel-axle" module.

图示

描述已自动生成

Fig. 3.7 Simulink Model for Wheel-Axle Module Configuration

* Set the size, weight, position, and other parameters of the chassis and support columns, and similarly implement a rigid connection to encapsulate them as a "Structure" module.

图示

描述已自动生成

Fig. 3.8 Simulink Model for Chassis and Support Structure Configuration

* Complete the connection between the body and wheel-axle modules, add a "Revolute Joint" module to allow the robot to rotate, and add a "Prismatic Joint" module to enable back-and-forth movement on the ground.
* Obtain the tilt angle from the "Revolute Joint" module, add a PID controller module to achieve closed-loop control of the entire system.

The overall model is as follows:

图示, 示意图

描述已自动生成

Fig. 3.9 Simulink Model for System Closed-Loop Control

The 3D model is as follows:

图形用户界面, 应用程序, Word

描述已自动生成

Fig. 3.10 3D Model of the Self-Balancing Robot System

1. **Balancing Robot Software Design**

**4.1 Operating System (FreeRTOS) Application**

For this balancing mobile robot project, which requires strong real-time performance, an immediate response is needed for balancing control, while other tasks, like OLED screen debugging information, can have slower response times. To maximize the CPU's responsiveness to critical tasks, we utilize the default integrated FreeRTOS operating system on the ESP32 development platform, eliminating the need for additional porting.

Given the resource limitations of microcontrollers like the ESP32, FreeRTOS's lightweight scheduling enables more efficient multitasking. FreeRTOS allows developers to create multiple tasks and assign priorities to them. Proper priority management helps the system allocate resources efficiently, preventing non-critical tasks from consuming too much time, thus improving the precision of the robot's control. Additionally, if new functionalities are needed, FreeRTOS makes it easy to add new task modules without affecting existing task logic.

In this design, three main tasks were implemented: the balance control task, the OLED display task, and the obstacle avoidance task. The balance control task has the highest priority, while the OLED task has the lowest priority. The stack size for each task is set to 2048 bytes.

**4.2 BLE Communication and Remote Control**

4.2.1 ESP32 Bluetooth Server Communication Setup

As the server, the ESP32's communication setup process is as Fig.4.1.

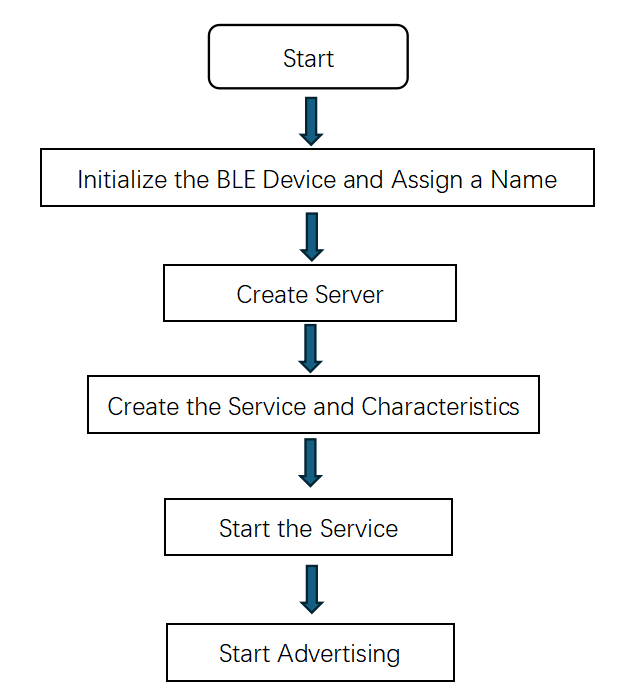


Fig. 4.1 Bluetooth Communication Server Creation Process

In the server creation process, we set a callback function for the BLE server named MyServerCallbacks, a custom class that inherits from BLEServerCallbacks. This class contains the logic for handling BLE device connections and disconnections. In the service creation step, we define the service UUID as "91bad492-b950-4226-aa2b-4ede9fa42f59", enabling clients to identify and connect to this service. For the characteristic creation, we define the UUID as "cba1d466-344c-4be3-ab3f-189f80dd7518" and assign a callback function MyCallbacks, which is also a custom class inheriting from BLECharacteristicCallbacks. It handles the reading and writing operations of the characteristic. When the mobile device sends data to the ESP32, onWrite() is triggered, and receivedValue retrieves the sent value. Based on the received data, it adjusts the motor encoder count values corresponding to actions like turning left, right, moving forward, backward, or stopping. Advertising is a BLE device mechanism to make itself discoverable by other devices (e.g., mobile phones). This step signals that the ESP32 starts broadcasting its BLE service, allowing it to be detected by mobile or other devices through scanning.

4.1.2 Bluetooth Remote Control App Design

We designed a mobile app for Bluetooth remote control, with its interface shown in Fig.4.2,

图表, 瀑布图

描述已自动生成

Fig. 4.2 Bluetooth remote control APP

In this APP, click "Start Scanning" to begin scanning for available Bluetooth devices. Click "Select Bluetooth Device" to open a list of available devices and complete the selection by clicking on the desired device. If the connection is successful, the status bar text changes from "Disconnected" to "Connected," and the "Disconnect" button becomes enabled. Clicking this button will disconnect the device, and the status bar text will revert to "Disconnected."

Once the Bluetooth connection is established, the five function buttons can be used to control the balancing mobile robot to perform the corresponding actions: "Move Forward," "Move Backward," "Turn Right," "Turn Left," and "Stop."

The code for visual programming of the APP is as Fig.4.3

日程表

描述已自动生成

Fig. 4.3 Code for visual programming of the APP

**4.3 Inclination Calculation Based on Kalman Filter and Motion Sensor MPU6050**

In this project, we use the MPU6050, a six-axis motion sensor with a 3-axis accelerometer and a 3-axis gyroscope, providing acceleration and angular velocity data for the three axes. Acceleration data can be used to calculate the tilt angle in static or low-speed situations, but its accuracy decreases in fast motion. Gyroscope data can track rapid angular changes but tends to accumulate significant drift error.

Fusion algorithms such as complementary filtering or Kalman filtering can combine the strengths of both data types, offering a more stable and accurate tilt estimation. Kalman filtering, though more complex than complementary filtering, provides higher accuracy. By combining the dynamic model and observation model, the Kalman filter continuously corrects the accelerometer and gyroscope data to achieve a more accurate tilt angle.

4.3.1 The Main Function Designs

Initialization function car\_imu\_init(): Initializes I2C communication, specifying the SDA and SCL pins and communication speed; initializes the MPU6050 sensor; uses the calcGyroOffsets() function to calibrate the gyroscope offsets to reduce system error; records the current time with the millis() function in imu\_fetch\_timer for subsequent interval calculations.

Time interval update function car\_imu\_update\_interval: Uses the millis() function to record the current time in current\_time; calculates the interval as current\_time - imu\_fetch\_timer and checks if it is zero. If zero, it manually sets it to 0.0001 to avoid division by zero errors in subsequent calculations; updates imu\_fetch\_timer to the current time.

Angle calculation function car\_imu\_compute\_angles(): First, it uses the update() function to retrieve the latest data from the MPU6050; calculates the tilt angle based on accelerometer data using the following formula:

Here, angle\_x and angle\_y are the tilt angles of the X and Y axes calculated from the accelerometer data, while a\_x, a\_y, and a\_z are the raw data directly obtained from the three axes of the accelerometer. The angular velocity is directly obtained from the gyroscope data. The tilt angles calculated from the accelerometer, the angular velocity from the gyroscope, and the time interval are passed to the Kalman filter function to complete the final tilt angle calculation.

4.3.2 Kalman Filter Function Design

* In this one-dimensional filter, the filter states are the angle and the gyroscope bias . In the state prediction phase, the angular velocity information from the gyroscope is used to update the angle, with the formula as follows:

Here, is the angular velocity measured by the gyroscope, is the bias of the angular velocity, and dt is the time increment.

* Next, the angle error is calculated using the following formula:

Where is the measured angle, is the predicted angle, and the error represents the observation residual, which is the difference between the measurement and the prediction.

* Next, the rate of change of the covariance matrix, , is calculated based on the process noise covariance of the angle noise, , and the process noise covariance of the gyroscope noise, . This predicted covariance rate is then used to update the covariance matrix .
* The Kalman gain is calculated to combine the predicted and measured values, with the formula as follows:

Where is the covariance matrix, is the observation matrix, and is the measurement noise.

* The covariance matrix is updated based on the calculated Kalman gain, with the formula as follows:
* Use the Kalman gain to correct the angle estimate angle and the bias .

**4.4 OLED Display**

Using the I2C communication protocol, the display is first cleared at startup, and the text color, size, and starting position are set. The encoder counts for the left and right motors and the Bluetooth connection status are displayed on the screen, updating every 20 ms.

**4.5 Motor Control**

Main function design is as follows:

Motor GPIO Initialization Functions motor\_l\_gpio\_init and motor\_r\_gpio\_init: Configure the two motor encoder pins as input mode, and set the PWM pin and two direction pins as output mode.

Motor PWM Setting Functions motor\_l\_set\_pwm and motor\_r\_set\_pwm: Use the constrain function to limit the speed within the range (-255, 255). If the desired speed is positive, the direction pin...

**4.6 Balance Control**  
4.6.1 Design of the PID Controllers for Balance, Speed, and Steering

First, three PID controllers are designed:

Upright loop PD control: The return value is as follows

Where is the current tilt angle, is the mechanical midpoint, and is the X-axis angular velocity.

Speed loop PI control: The input parameters include the target speed, current speed, and speed integral term.

First, limit the integral term to prevent it from becoming too large. If the integral term is greater than 700 or less than -700, it is limited to 50. The calculation result is as follows:

Where Speed is the current speed, Speed\_target is the target speed, Speed\_integral is the speed integral term, is the proportional coefficient of the speed loop, and is the integral coefficient of the speed loop.

Steering control uses a P controller. The input parameters are the left and right motor positions. The error is calculated as the difference between the left and right motor positions. When the error is between (-30, 30), the return value is set to:

4.6.2 Balance Control Task Program Flow

In the balance control task, the program flow is shown in Fig.6.4.

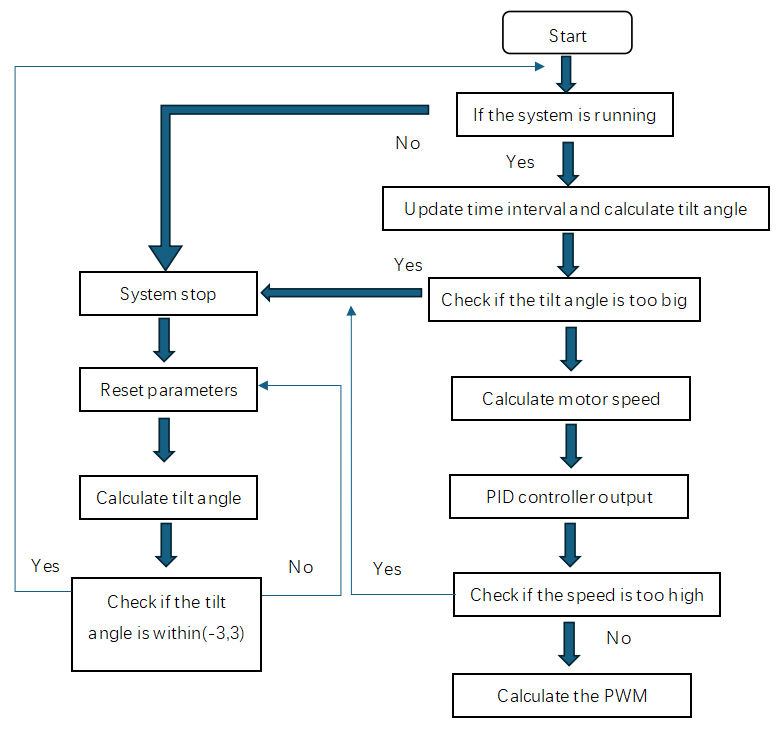


Fig. 4.4 Balance control program flow

In the balance control task, the program flow includes checks to determine whether the tilt angle is greater than 50° or less than -50° and whether the speed is too high. These checks are designed to prevent the balancing mobile robot from losing control and to initiate power-off protection.

During the motor speed calculation, the current encoder count and the previous count are used to estimate the speed. Then, filtering is applied, and a weighted average is used for smoothing to obtain the filtered speed.

The PID controller outputs include the results from the upright loop, speed loop, and steering loop. When calculating PWM, for the left motor, the output is the upright loop minus the speed loop plus the steering loop, and for the right motor, it is the upright loop minus the speed loop minus the steering loop. The difference in speed achieves steering control.

When the system stops, the parameters are reset, including PWM, encoders, PID controller outputs, and speed. After the system stops, a delay loop is initiated to wait for the system to automatically restart in an appropriate state. During each cycle, the time interval and tilt angle are calculated. When the tilt angle is between (-3, 3), indicating the system is close to upright, it will automatically restart.

**4.7 Main program flow**

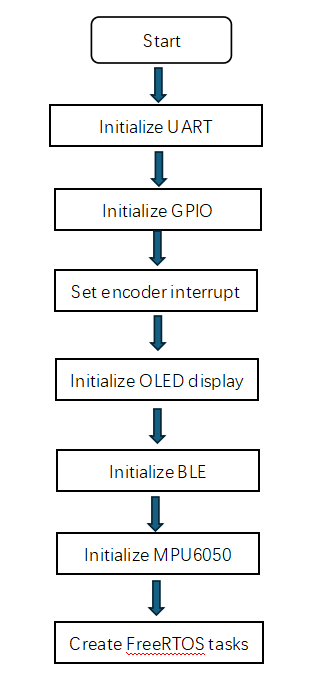


Fig. 4.5 Main program flow

Interrupts are set for the encoder signal pins of the left and right motors. When the encoder detects a rising edge (RISING) of the signal, the corresponding interrupt service function is called. FreeRTOS tasks are created, including balance control, OLED display, ultrasonic obstacle avoidance, and others.

**4. Schedule**

|  |  |  |
| --- | --- | --- |
| Summary Milestones | Planned Milestone Date | Actual Milestone Date |
| Requirement analysis and completion of the project design document | 30/08/2024 | 30/08/2024 |
| Hardware platform assembly and power-on debugging | 20/09/2024 | 20/09/2024 |
| MPU6050 attitude calculation and motor driver implementation | 04/10/2024 | 26/09/2024 |
| Control algorithm implementation and testing for pid, lqr, etc. | 25/10/2024 | 17/10/2024 |
| Implementation of additional functions, such as bluetooth remote control and ultrasonic range and obstacle avoidance | 08/11/2024 | 05/11/2024 |
| Functional integration testing and acceptance | 15/11/2024 | 13//11/2024 |

**5. Cost (if any)**

|  |  |  |
| --- | --- | --- |
| Cost Items | Planned Costs | Actual Costs |
| ESP32S3 Dev Kit | 20 | 7.16 |
| Power Bank | 50 | 33.13 |
| Supersonic Sensor | 5 | 1.57 |
| Car chassis | 50 | 25.55 |
| Electronic Components | 30 | 43.89 |
| IMU Module | 10 | 4.04 |
| Screws | 10 | 6.83 |
| Project Total Costs | 175 | 122.17 |

**6. Outcomes / Benefits**

**Outcomes**

* Successfully designed and implemented a self-balancing car capable of utilizing sensor feedback to detect body inclination. Through the application of the PID control algorithm, the system dynamically adjusts motor output, allowing the car to automatically recover from tilts and maintain balance.
* Integrated the MPU6050 attitude sensor, HC-SR04 ultrasonic sensor, motor driver module, and ESP32 microcontroller to establish a complete embedded control system with essential functions, including obstacle detection, position stabilization, and remote control capabilities.
* Developed an inverted pendulum model of the car and conducted a comprehensive kinematic analysis. This analysis served as a foundational tool for understanding the car’s dynamics and informed subsequent PID parameter tuning, enhancing control accuracy and system stability.
* Utilized Simulink to perform control simulations and establish a control system model. This simulation environment allowed for the adjustment and fine-tuning of PID parameters in a controlled setting, providing an efficient method to analyze system behavior and optimize performance before implementing on the actual hardware. The simulation approach offered significant advantages, including the ability to identify potential control issues, reduce trial-and-error on physical components, and facilitate rapid prototyping and testing of different control strategies.
* Developed and implemented a closed-loop control system that leverages real-time sensor feedback and the PID control algorithm, enabling the car to dynamically adapt its posture and respond to disturbances, achieving real-time balance.
* Conducted multiple tests and optimizations on PID parameters, which improved the system’s responsiveness and stability, ensuring that the car can maintain effective balance while moving smoothly.
* Successfully developed and incorporated a Bluetooth module, enabling remote control via Bluetooth. This allows users to control the car wirelessly using a smartphone or computer, providing enhanced flexibility and operational versatility for the project.

**Benefits**

* Mastery of PID Control and Parameter Tuning

We gained a solid understanding of PID control and various tuning methods, including manual tuning, simulations in Simulink, and mathematical model analysis. This experience helped us see how different parameter adjustments impact system stability and responsiveness, strengthening our control algorithm skills.

* Expanded Hardware Knowledge

We learned the structure, functions, and working principles of key components such as the MPU6050 sensor, HC-SR04 ultrasonic module, motor drivers, and the ESP32 microcontroller. Through hands-on assembly and layout, we improved our skills in component integration, wiring organization, and interference reduction.

* Simulation and Modeling for Optimization

Using Simulink, we created a simulation model that enabled us to optimize PID parameters efficiently. This approach allowed us to test system behavior before physical implementation, saving time and improving accuracy.

* Practical and System Integration Skills

The project enhanced our hands-on abilities in hardware setup, sensor calibration, and system integration. We learned how to optimize layouts and achieve stable performance in complex setups.

* Strengthened Team Collaboration

This project emphasized the importance of teamwork. By leveraging each member’s strengths, we improved efficiency and developed strong communication and collaboration skills that will be valuable in future projects.

**7. Individual Reports from Team Members**

* **Hou Yabin**

In this project, I was mainly responsible for the design of the algorithm, the construction of the system simulation model, the preliminary tuning of PID parameters and the writing of the driver program of the ultrasonic module.

This project uses PID control algorithm to control the tilt Angle, position and steering Angle of the car, and adopts three parallel PD controller, PI controller and P controller to realize it. Different from the common cascade system in textbooks, the program implementation of the parallel level system is more convenient. This project made me realize that there is a big difference between the knowledge in the textbook and the project practice. Most of the theories in the textbook are conclusions and reasoning under ideal conditions, and there are many interfering factors that need to be considered if they are to be applied in the actual project.

The establishment of the simulation model depends on the physical modeling of the team members, and on this basis, the state space model of the system is derived, and then the simulation model is built on Simulink to provide conditions for the initial tuning of PID controller parameters. At the beginning, the state space model of the car could not be established, which made it very difficult to carry out the task. In the control course, the state space model is usually obtained by the kinematic model, but in the actual project, the kinematic model of the car is very difficult to build, such as the mass M, the position of the center of mass l, and other parameters are difficult to measure accurately. This makes me once again realize the gap between theoretical knowledge and practical projects, and I will pay more attention to combining practice in future study.

I also participated in PID parameter setting, which was divided into two parts: simulation debugging and hardware debugging. The parameters of simulation debugging are roughly selected to shorten the range of hardware debugging, and hardware debugging is fine-tuned on the basis of simulation debugging to offset various interference factors ignored in the simulation process. In theory, I think this is a reasonable project process, but in actual work, the construction of simulation models is carried out simultaneously with hardware debugging, which makes the project advance faster and the communication between team members more frequent. I can't help but think that in real projects, it may not be necessary to strictly follow the planned flow, or that many seemingly serial tasks can actually be considered in parallel.

I also completed the writing of the driver program of ultrasonic module. Most of my undergraduate courses were biased towards software. The combination of hardware and software brought me a great challenge, but also a great harvest. The writing of the ultrasonic module driver is not as difficult as I imagined, and the so-called hardware and software combination is only to treat the pin interface as a variable in the program. In team tasks, normative programming standards and communication become particularly important. When writing driver modules, we should consider the simplicity of program interfaces and code legibility. At the same time, we should also read existing code, observe variable declaration habits and pin declaration positions, and ensure the consistency of the overall program as much as possible. Therefore, it is easy to make integrated member calls and modifications.

In fact, I also wrote a simple obstacle avoidance algorithm, but due to some problems in the program framework and time constraints, I did not come and put it into practice, which is a little regrettable. The main problem of this result should have been taken into account when the program framework was built, but I just knew little about it and did not pay enough attention to it at that time. This is a lesson that I will pay extra attention to in my future study and work.

* **Huang Jian**

In this EE6008 project, I designed the hardware and embedded software for this vehicle, derived its dynamics model, and conducted simulations using MATLAB. It can be observed that the MATLAB simulation results align well with the vehicle's dynamics model.

Transitioning from the physical world to code requires some clever techniques to establish the connection between these two systems. We measured the relationship between the motor output torque and the vehicle weight to obtain this link, which made our simulation and debugging work smoother.

During debugging, a good theoretical foundation is essential, but having a deep intuitive understanding of the theory is even more critical. Simply facing mathematical theory does not significantly aid debugging, but once we have a profound intuitive grasp of the theory, the debugging process becomes much easier. We can understand how changes to parameters will impact the physical world, rather than merely noting whether our mathematical model converges or diverges.

Of course, this model has many limitations that are not reflected in the mathematical derivation. For example, the motor has static friction; it will not run when the PWM value is below a certain level. The PWM driver also needs dead-time control and speed control feedback—too fast feedback speed may burn out the driver and pose a greater challenge to the power supply. These limitations add difficulty to the debugging process, but we ultimately overcame them.

This project allowed me to complete the entire process from deriving the mathematical model to building the actual car, giving me insights into the gap between theory and practice and how to bridge them, thus enhancing my practical application skills.

* **Huang Xiaoying**

In the project "A Balancing Mobile Robot," my main responsibility was developing the dynamic model and contributing to the control algorithm, specifically for the PID control program.

For modeling, I treated the robot as an inverted pendulum system to capture its fundamental dynamics. To accurately describe these dynamics, I measured essential parameters like the robot's mass and the height of its center of mass. Using these parameters, I derived the state-space model with matrices A, B, C, and D, which allowed us to analyze the system’s stability, controllability, and observability. This model served as the basis for designing and optimizing control strategies.

Given the complex nature of the system, I made several reasonable simplifying assumptions to keep the model focused on core dynamics. For instance, I assumed linear motion without lateral disturbances and started with the initial state at rest. Additionally, I incorporated damping factors to account for practical resistive forces, such as friction, to improve the model's accuracy. Despite these simplifications, the model effectively revealed that the system would be naturally unstable without control intervention, underscoring the need for an active control system to maintain balance.

I also contributed to developing part of the PID control algorithm, using a dual-loop approach with separate control for the robot’s speed and balance. This dual-loop design was necessary to manage both the position and angle effectively, providing faster response and stability in the system. The inner loop focused on stabilizing the tilt angle, while the outer loop controlled the speed, creating a robust layered control structure.

One of the primary challenges I encountered was achieving accurate modeling and parameter tuning. Due to the inherent approximations in the model, including estimated or simplified physical parameters, it was challenging to directly correlate control parameters, such as the PWM values, with the physical behavior of the robot. These approximations made it difficult to fine-tune the PID parameters for the desired performance directly based on the model.

To address this, I adopted a strategy involving pole-zero placement and root locus analysis to determine initial parameters for the PID controller. After establishing these initial values, I iteratively refined them through experimental adjustments. In each iteration, I used the tuned parameters to infer adjustments in physical parameters, which allowed me to update and improve the model iteratively. This approach, while unconventional, proved effective in bridging the gap between the theoretical model and the practical behavior of the robot, improving both the model's accuracy and the control system’s performance.

This project was an excellent opportunity to apply the theoretical knowledge I had gained in class to a real-world engineering task. Unlike solving textbook problems, where we often focus on finding the correct answer, this project required a deeper understanding of how each piece of data was derived. From measuring physical parameters to developing the dynamic model, deriving the state-space equations, coding the PID control algorithm, and finally seeing the balanced robot in action, each step was grounded in real-world principles and built upon the previous one. It was an experience of moving from theory to practical implementation, where each part had a traceable, logical progression.

I also realized that teamwork involves more than just dividing tasks based on individual strengths; effective collaboration requires clear communication and integration across all parts of the project. Without a comprehensive understanding of the project as a whole, it would have been challenging to coordinate and communicate effectively with my team members. Each step was interconnected, not isolated. For example, elements like code integration, hardware setup, control algorithms, and filtering needed to be seamlessly combined in a single final file. This project taught me that understanding the overall framework is essential for successfully executing my part.

* **Ju Jialin**

As the team leader, I was primarily responsible for scheduling, task allocation, and regular communication and reporting with our supervisor to coordinate the team’s work. In addition, on the technical side, I was mainly responsible for calculating the vehicle’s tilt angle using the Kalman filter and six-axis sensor MPU6050, implementing Bluetooth communication, creating Simscape 3D models, and developing a Bluetooth remote control mobile app using MIT App Inventor. I also contributed to the overall project design, physical modeling, and control algorithm development.

In this project, I gained comprehensive experience in both technical and management aspects. I came to deeply understand the importance of full-process management—from requirements analysis and design planning, to team division of labor, results integration, and final implementation.

At the beginning of the project, unfamiliarity with team members' technical capabilities led to idealistic task allocation, resulting in difficulties during code integration and delays in development. However, as the project progressed, we focused more on open discussions, set clear weekly task schedules, and established deadlines, which significantly improved development efficiency. Throughout this process, I not only enhanced my abilities in team progress management and resource allocation but also improved my attention to project management details. In terms of scheduling, I learned how to distribute tasks effectively to ensure each module progressed in sync, allowing us to keep the project on track. Regularly communicating with our supervisor, reporting on progress and challenges, receiving feedback, and making timely adjustments to our approach all played a crucial role in the successful completion of the project.

From a technical perspective, many of my personal achievements in this project involved breakthroughs from scratch. This was my first time working with the ESP32 and Arduino development platform, my first experience as a leader in a small team, and my first time using Simscape and MIT App Inventor as development tools.

Throughout this process, I encountered numerous challenges, and by continually consulting resources and testing different approaches, I achieved satisfactory results. One key area was the choice of filtering algorithms: initially, I used a simpler complementary filter, but later opted for the more complex Kalman filter to achieve higher accuracy. Applying its theory to this project deepened my understanding of its prediction and update steps, as well as the role of covariance matrices. This experience not only familiarized me with using the Kalman filter for tilt angle calculation but also helped me gain debugging experience.

Additionally, implementing Bluetooth communication required me to learn from scratch on an unfamiliar development platform. I gradually acquired the knowledge needed to synchronize the software design on both the ESP32 and the mobile device, ultimately achieving stable connectivity and communication. Due to my initial lack of familiarity with the ESP32, the app was first designed using Bluetooth Classic mode. After further research, I found that ESP32’s integrated BLE was more convenient, leading to a significant code overhaul. This experience strengthened my ability in hardware-software integration and showed me that every step, from protocol debugging to optimizing data transmission stability, requires detailed attention and fine-tuning.

* **LIU ZHAOYUE**

In this project, I was responsible for designing the hardware schematic of the self-balancing car and integrating key modules, including the ESP32 microcontroller, MPU6050 attitude sensor, motor driver module, and HC-SR04 ultrasonic sensor. I ensured compatibility between each component and the ESP32’s GPIO pins and communication protocols, and I optimized the power layout to enhance system stability. My contributions ensured stable data acquisition from the sensors and precise motor control, effectively enabling the car’s balance control and obstacle avoidance functions.

Since my undergraduate major is mechanical engineering, and this project is more control-oriented, I had to learn a variety of new concepts.

Firstly, in hardware circuitry, I learned about selecting suitable components, such as small motors and the MPU6050 attitude sensor, while also focusing on circuit layout and connection stability. For example, the continuous acceleration and deceleration of the car during operation impose demands on the vibration resistance and connection robustness of the circuitry. I learned how to enhance circuit reliability through proper layout and secure component mounting.

Secondly, I studied control system design, gaining an understanding of the fundamentals of open-loop and closed-loop control, especially feedback control in closed-loop systems. In this project, we implemented a closed-loop control system incorporating PI, PD, and P controllers. Using the attitude sensor, the system detects the car's tilt angle, position deviation, and speed difference between the left and right motors. Based on this feedback, the controller dynamically adjusts motor output to maintain the car’s balance.

Additionally, I learned about signal processing, specifically the principles and application of Kalman filtering. By integrating Kalman filtering into the attitude estimation algorithm, we effectively reduced noise interference, minimizing errors caused by mechanical vibrations, gyro drift, and other disturbances during the car's operation.

In the early stages of hardware design for the balance car, we encountered several challenges in component selection and procurement. Due to the high cost of certain components domestically and in Singapore, we opted to source suitable alternatives from China. Additionally, we needed to strengthen our theoretical knowledge in areas like the input and output characteristics of components such as the ESP32 and MPU6050, as well as fundamental concepts of filtering and level shifting.

Since our power source is a portable charger, which typically outputs 5V, and modules like the ESP32 and MPU6050 operate at 3.3V, we included a voltage regulator to step down the 5V to a stable 3.3V. To ensure stable power supply, we also added decoupling capacitors at the input to filter out high-frequency noise.

During testing and troubleshooting, we encountered several issues:

Firstly, there was a mismatch between hardware components and software configuration. The initial schematic and code setup required multiple adjustments to the I/O configuration and control system to ensure smooth operation.

Secondly, during adjustments and testing, some component connections were unstable, leading to shifts in the car’s center of gravity, which affected the balance control of the inverted pendulum model. To address this, we redesigned the hardware assembly structure and tested Kalman filtering to effectively reduce unnecessary vibration noise, successfully resolving the related hardware issues.

**8. References**

[1] J. Dabbagh and I. H. Altas, "Nonlinear Two-Wheeled Self-Balancing Robot Control Using LQR and LQG Controllers," 2019 11th International Conference on Electrical and Electronics Engineering (ELECO), Bursa, Turkey, 2019, pp. 855-859, doi: 10.23919/ELECO47770.2019.8990610.

[2] Dabbagh, J., & Altas, I. H. (2019). Nonlinear two-wheeled self-balancing robot control using LQR and LQG controllers. 2019 11th International Conference on Electrical and Electronics Engineering (ELECO), Bursa, Turkey, 855-859. https://doi.org/10.23919/ELECO47770.2019.8990610

[3] Mohamed Gad, O. M., Saleh, S. Z. M., Bulbul, M. A., & Khadraoui, S. (2022). Design and control of two wheeled self-balancing robot (TWSBR). 2022 Advances in Science and Engineering Technology International Conferences (ASET), Dubai, United Arab Emirates, 1-6. https://doi.org/10.1109/ASET53988.2022.9735004

[4] Petcu, R., & Andreescu, G.-D. (2022). Two-wheeled inverted-pendulum self-balancing robot with LQR or PID control. 2022 IEEE 10th Jubilee International Conference on Computational Cybernetics and Cyber-Medical Systems (ICCC), Reykjavík, Iceland, 345-350. https://doi.org/10.1109/ICCC202255925.2022.9922862

[5] Qiu, J., et al. (2017). Two-wheeled self-balancing robot modeling and nonlinear control. 2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Jeju, South Korea, 734-739. https://doi.org/10.1109/URAI.2017.7992813

[6] Zimit, A. Y., Yap, H. J., Hamza, M. F., Siradjuddin, I., Hendrik, B., & Herawan, T. (2018). Modelling and experimental analysis of a two-wheeled self-balance robot using PID controller. In Gervasi, O., et al. (Eds.), Computational Science and Its Applications – ICCSA 2018. Lecture Notes in Computer Science, vol. 10961, Springer, Cham. <https://doi.org/10.1007/978-3-319-95165-2_48>

[7] Díaz-Téllez, J., Gutierrez-Vicente, V., Estevez-Carreon, J., Ramírez-Cárdenas, O. D., & García-Ramirez, R. S. (2021). Nonlinear control of a two-wheeled self-balancing autonomous mobile robot. In Batyrshin, I., Gelbukh, A., & Sidorov, G. (Eds.), Advances in Soft Computing. MICAI 2021. Lecture Notes in Computer Science, vol. 13068, Springer, Cham. <https://doi.org/10.1007/978-3-030-89820-5_28>

[8] Imtiaz, M. A., Naveed, M., Bibi, N., Aziz, S., & Naqvi, S. Z. H. (2018). Control system design, analysis & implementation of two wheeled self-balancing robot (TWSBR). 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 431-437. https://doi.org/10.1109/IEMCON.2018.8614858

·

**Appendix -** Project Members Information

|  |  |  |  |
| --- | --- | --- | --- |
|  | Name | Project contributions | Report Contribution |
| 1 | Ju Jialin | Team Leader；Simscape 3D modeling; Bluetooth remote control mobile App design; Inclination calculation based on kalman filter and motion sensor MPU6050; Bluetooth communication implementation | Pages 3, 17-26, 29-30 |
| 2 | Huang Xiaoying | Kinetic model, incorporating state-space representation for precise stability and control analysis; PID control implementation | Pages 2, 6-12, 37 |
| 3 | Huang Jian | ESP32 control program; Kinetic Model; Simulink simulation model of control system; low-level hardware driver; hardware design and fabricate | Pages 4-10,19,23-25,28 |
| 4 | Liu Zhaoyue | Hardware design and integration; System block diagram drawing; Schematic analysis and drawing; Auxiliary PID algorithm parameter adjustment | Pages 4-6 |
| 5 | Hou Yabin | Simulink simulation model of control system,control algorithm design， preliminary tuning of PID parameters, ultrasonic module driver | Pages 12-16 |