**A Balancing Mobile Robot**

Huang Jian

# Dynamic Analysis

## Wheels Analysis

* Motor output torque:
* Wheel mass:
* Wheel radius:
* Wheel displacement:
* Wheel angular velocity:
* Frictional force exerted by the ground on the wheel:
* Horizontal force exerted by the vehicle body on the wheel:
* Vertical force exerted by the vehicle body on the wheel:

Perform an analysis on the wheel, set up the differential equations, and eliminate

Substitute the relationship between angular velocity and displacement  
It can be obtained:

Since the weight of the plastic wheel is light enough to be neglected, we can obtain:

Since the controller outputs the motor torque as a PWM value, we define:  
So the final equation is:

## Car Body Analysis

* Mass of car body:
* Distance from the center of mass to the wheel axle:
* Angle of offset from the neutral position:

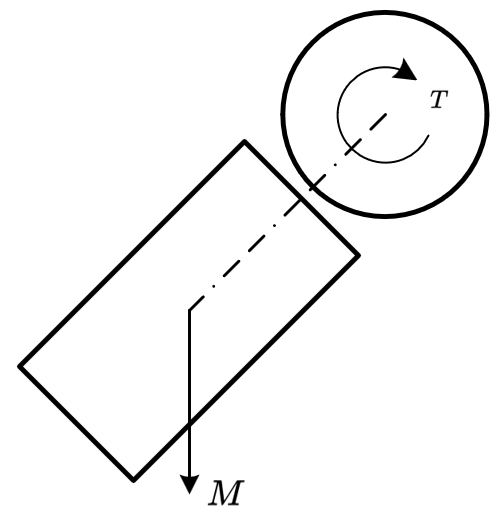
Perform analysis on the car body:

By combining the above equations, we obtain:  
Near the neutral position, where , we can obtain:  
Substitute into the system of equations and linearize, we obtain:

We obtain the state-space equations:

## Obtain Coefficient

We invert the car, fix the wheel, and measure the PWM value when the car swings to 45°. The measured value is, , , , so we can obtain:  
So



**Obtain Coefficient**

The state-space equations changes to:

# Feedback Control and Matlab Simulation

The basic control loop contains a vertical control loop and a speed control loop, which outputs PWM value to control the torque of the motors.

## Vertical Control Loop

The vertical control loop includes a PD controller, which is used to keep the car standing vertically.

To implement the theoretical controller into a practical program, the derivative of the angle can be replaced with the IMU's gyro data. Thus, the actual PD controller equation is:

The actual code is:

return vertical\_kp \* (angle\_x - vertical\_mid) + vertical\_kd \* gyro\_rate\_x;

## Speed Control Loop

The speed control loop includes a PI controller which is used to eliminate one of the two steady statics of vertical control loop.

It gives a positive feedback to invert the direction of the car to change speed direction. To implement the theoretical controller into a practical program, the integration of speed can be replaced with the encoder's count (position). Thus, the actual PI controller is:

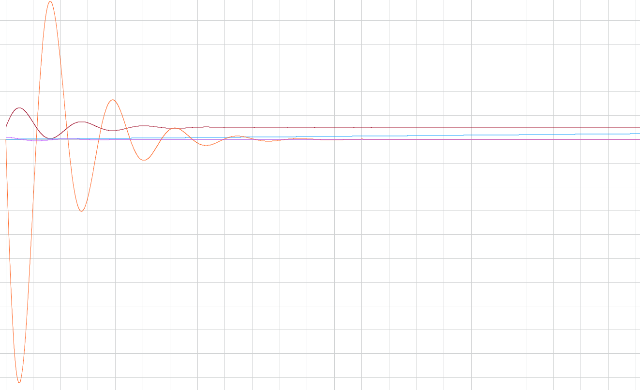
The actual code is:

return (speed - target\_speed) \* speed\_kp + speed\_intergral \* speed\_ki;

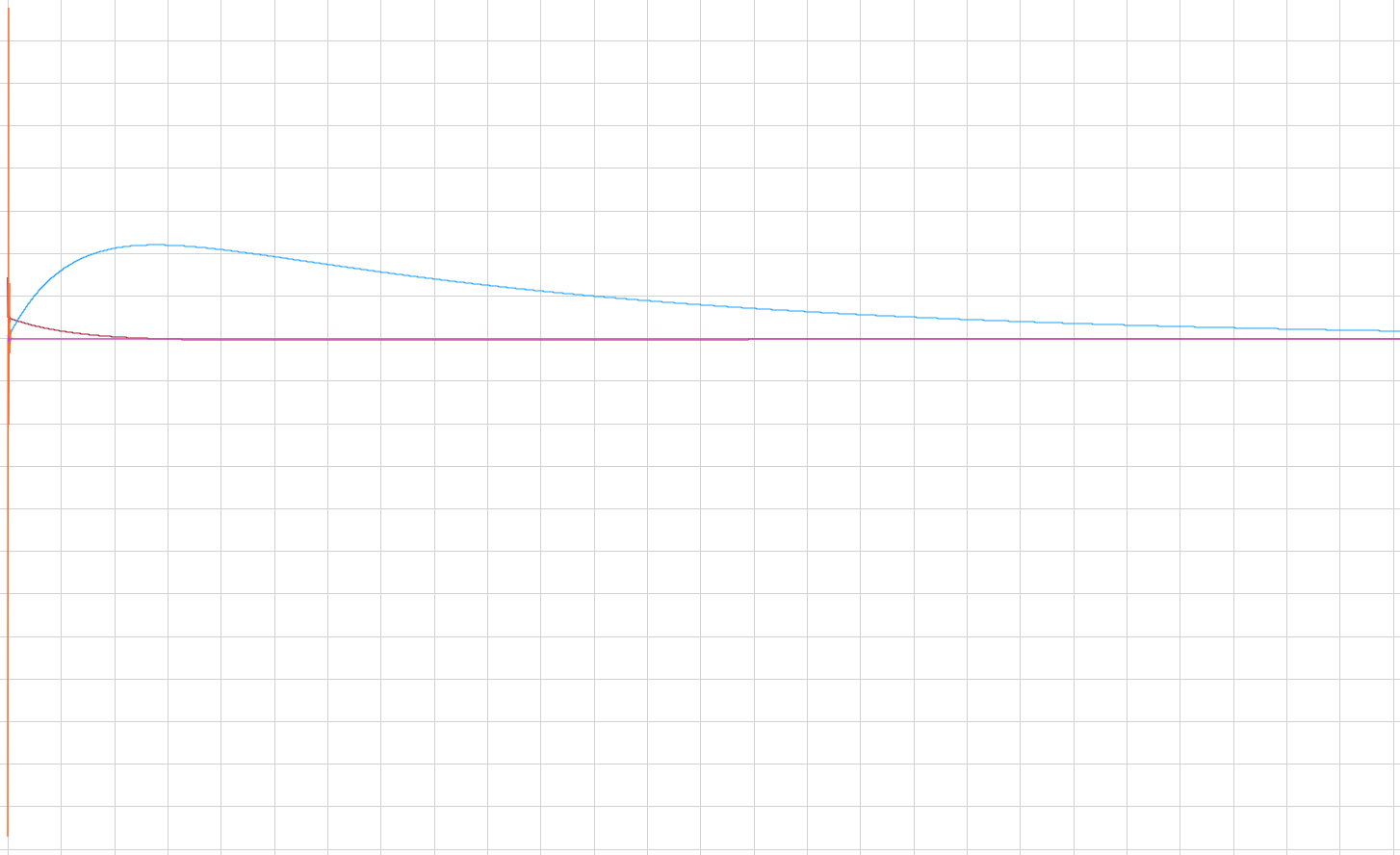
## Final Control Loop & Matlab Simulation

The final control algriothm is

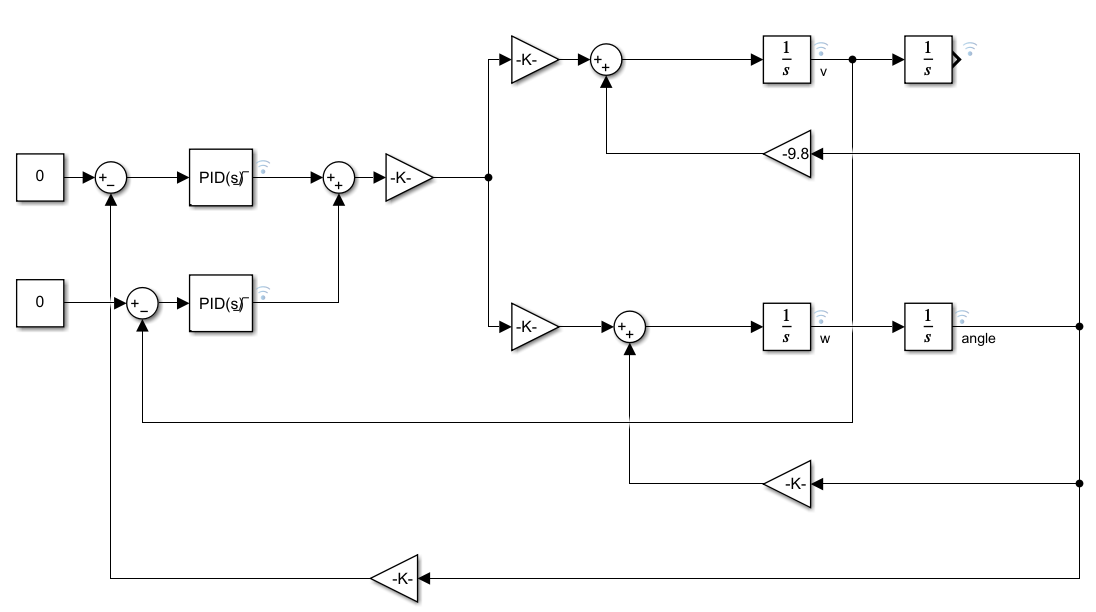
Matlab simulations for vertical control and speed control loop work well with the dynamic model which we obtained above and the actual control program parameters. The result shows that the car will continue moving without speed control loop, because we have two stable state for car to stay standing. Once the speed control loop was added, the position and speed converge and finally the car will stay at its original position.



**Vertical Control Loop Simulation Result**



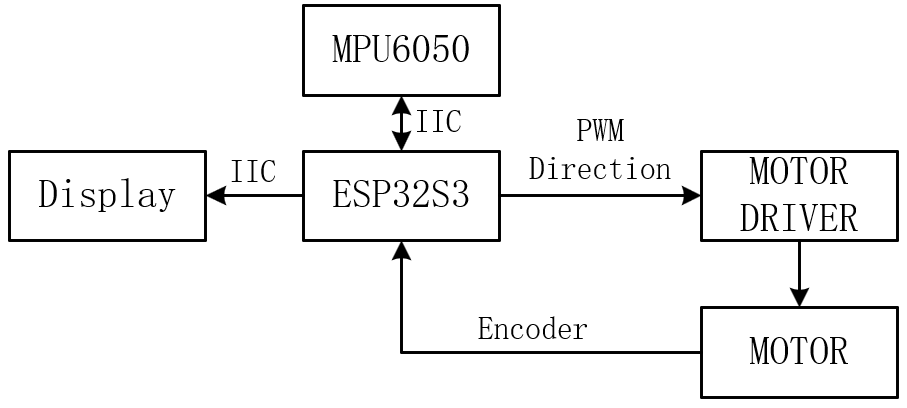
**Vertical and Verlocity control loop result**



**Simulink Module**

After Matlab simulation ,we can find that

# Hardware Design



**Main Hardware Design**

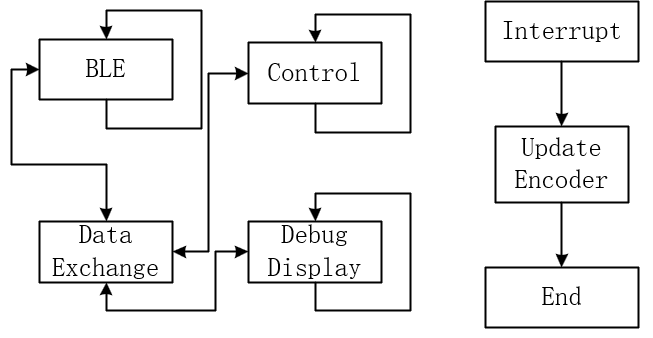
To control the whole system, I used an ESP32S3 module to run an Arduino platform. ESP32S3 module has two powerful RISC-V Xtensa cores running at 240MHz so it is enough for a balancing mobile robot.

MPU6050 is chosen for the IMU part. It is connected to ESP32S3 module via IIC bus and can provide a maximun speed of 400kbps.

The power system uses a DC motor with an encoder. This type of motor can be driven by PWM, which is a simple driving method where the output torque is proportional to the PWM duty cycle of the driver, making it ideal for balance car applications.

The power supply uses a PD-powered power bank and provides a 12V voltage to the motor driver board through a USB Type-C adapter. The motor driver board uses DC-DC to step down the voltage to supply the ESP32. To ensure stable power supply, a large capacitor is connected in parallel to provide the current needed for the driver's jitter around zero point.

# Program Architecture



**Program Architecture**

The program uses FreeRTOS for multithreading management. The core control program and sensor sampling program run on core 1, while the rest, including Bluetooth communication, debugging communication, and the Arduino base library, run on core 0.

# Reflection

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.