Study on Ziegler–Nichols PID Tuning based on a Two-Wheeled Self-Balancing Robot

Yin Si-kai, Zhang Qing-bo, Yang Deng-feng

***Abstract*—This paper presents a student project on the control of a two-wheel self-balancing robot. The robot is driven by two DC motors using a PID controller, and is equipped with a STM32F103 microcontroller, a three-axis gyroscope and a 2-axis accelerometer for attitude control. The tuning of PID control parameters utilizes the Ziegler-Nichols Tuning method to optimize the operation performance of target robot. The testing result shows that this self-balancing system performs much better after parameter tuning. The further work on alternative tuning method including DAP tuning is also discussed.**

***Index Terms*—PID tuning, Self-balancing, Ziegler–Nichols method, Signal processing**

I. Introduction

IN the past decades, mobile robots have entered the civilian and personal spaces such as hospitals, schools and ordinary homes. Based on these commercial product, scientists have been conducting researches on urban transport non-pollutant vehicles. Self-balancing robots are kind of robot which can provide the purposes proposed including high energy efficiency, great maneuverability without fossil fuel consumption. Segway robot is the first self-balancing robot which was invented by Dean Kamen in 2001 as the 'first self-balancing, electric-powered transportation device'. Segway is the implementation model of the inverted pendulum problem algorithm. Based on the PID controller, microcontroller with programming and extra Kalman filter, the system equilibrium could be maintained with external sensors and DC motors.

In this paper, we report a course project on the control of a two-wheel self-balancing robot. The robot is driven by two DC motors, and is equipped with a development board which is based on the STM32F103 microcontroller, a three-axis gyroscope and a 2-axis accelerometer for attitude determination. To compensate for gyro drifts problem in COTS sensors, a complementary filter is implemented in attitude acquisition process. Further, the proportional-integral-differential(PID) controller is implemented in control system. As there is no determinant and robust scheme on controlling the dynamic mechanic system embedded with microcontroller units, the Ziegler–Nichols method is implemented. After imposing heavy loads on the target balancing robot, the optimization effect of PID tuning could be revealed from the system's improved performance. The sensor data is extracted from serial port and signal processing and data mining analysis could be implemented to conclude the overall tuning experience on self-balancing system.

II. Self-balancing theory and PID control

A. Main Elements of the Self-balancing System

The structure of a self-balancing robot can be classified into three parts: sensors, motor and motor control, and develop board. To balance the robot, data of the robot's tilt angle and angular velocity are needed. Using MEMS sensors including a gyroscope, an accelerometer and wheel-angle encoders, all the data needed for balancing control can be measured.

Motor selection for the balancing robot emphasizes torque output instead of velocity, because it has to oppose the rotational moment that gravity applies on the robot. For our target system, a pair of the Pololu 67:1 Metal Gearmotor 37Dx54L mm with the above 64 CPR Encoder is included, and all the following tuning calculations must be based on this.

The system architecture of the self-balancing robot is as shown below, where the forward loop comprises the robot, a balancing controller which delivers a motor-control signal, and a inner-loop controller for wheel synchronization. Feedback is provided through a complementary filter whose function is to provide an estimate of the robot tilt angle from gyroscope and accelerometer measurements. In order to obtain the appropriate driving signal for motor control, the balancing controller must be implemented to the robot.

Parameter	**Definition**	**Value**	**Unit**
M_p	Mass of the robot's chassis	1.37	Kg
I_p	Moment of inertia of the robot's chassis	0.0074	$Kg \cdot m^2$
l	Distance between wheel center and robot's gravity center	0.0841	m
M_w	Mass of the robot's wheel	0.08	Kg

k_m	Motor's torque constant	0.2825	$N \cdot m / A$
k_e	Back EMF constant	0.67859	$V \cdot s / rad$
R	Terminal Resistance	9.124	Ω
r	Wheel radius	0.0334	m

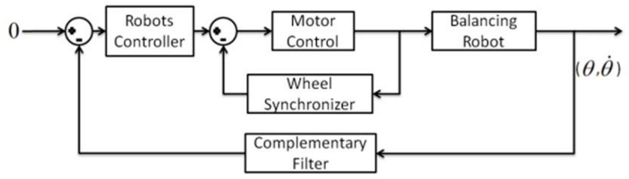

Fig. 1. Basic Structure of the Self-balancing System

B. *Control Strategy*

In the field of industrial automation, Proportional-Integral-Derivative (PID) control is the most common control algorithm used in industry and has been universally accepted in industrial control. The popularity of PID controllers can be attributed partly to their robust performance in a wide range of operating conditions and partly to their functional simplicity, which allows engineers to operate them in a simple, straightforward manner.

As the name suggests, PID algorithm consists of three basic coefficients; proportional P, integral I and derivative D, which are varied to get optimal response. PID controller performs basically on the comparison and compensation on the output value against set point (ideal value). The control block diagram of PID controller could be illustrated by the following chart.

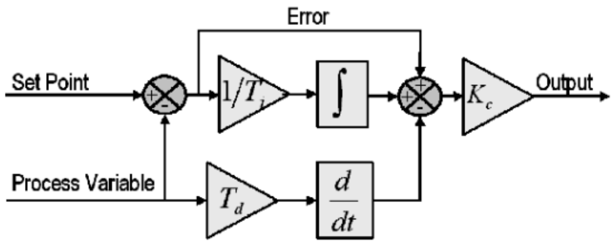

Fig. 2. Basic Structure of PID Control

After implementing the PID controller, the system response varies with external conditions and systematic features. Due to the inherent weakness of this simple controller, the control effect could not be perfect for all systems. A typical system response with close-loop PID controller is shown as below.

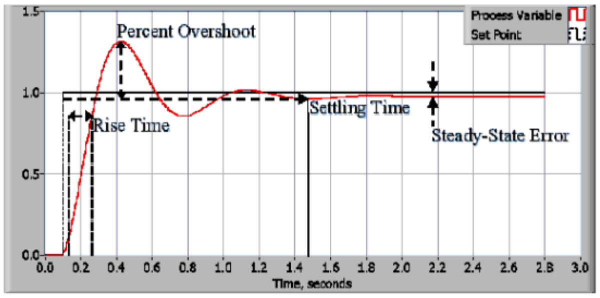

Fig. 3. PID Control System Response

In order to optimize the system performance, certain compromise and balance must be achieved between rise time, overshoot, settling time and steady-state error, which are four crucial parameters in determining the system transient performance. The step is usually called PID tuning, which could take engineering hours or even days to realize.

In regard with PID control theory, the proportional component depends only on the difference between the set point and the process variable. This difference is referred to as the Error term. The proportional gain (Kc) determines the ratio of output response to the error signal. In general, increasing the proportional gain will increase the speed of the control system response. However, if the proportional gain is too large, the process variable will begin to oscillate. If Kc is increased further, the oscillations will become larger and the system will become unstable and may even oscillate out of control.

The integral component sums the error term over time. The result is that even a small error term will cause the integral component to increase slowly. The integral response will continually increase over time unless the error is zero, so the effect is to drive the Steady-State error to zero. Steady-State error is the final difference between the process variable and set point. A phenomenon called integral windup results when integral action saturates a controller without the controller driving the error signal toward zero.

The derivative component causes the output to decrease if the process variable is increasing rapidly. The derivative response is proportional to the rate of change of the process variable. Increasing the derivative time (Td) parameter will cause the control system to react more strongly to changes in the error term and will increase the speed of the overall control system response. Most practical control systems use very small derivative time (Td), because the Derivative Response is highly sensitive to noise in the process variable signal. If the sensor feedback signal is noisy or if the control loop rate is too slow, the derivative response can make the control system unstable.

To optimize the performance of our dynamic self-balancing robot, we set out to conducting PID tuning based on Ziegler–Nichols PID tuning method, which is a really typical measure to acquire the appropriate parameter value.

III. Systematic PID tuning based on Ziegler–Nichols Method

The Ziegler–Nichols tuning method is a heuristic method of tuning a PID controller. It was developed by John G. Ziegler and Nathaniel B. Nichols. This tuning is performed by setting the I (integral) and D (derivative) gains to zero. The "P" (proportional) gain is then increased (from zero) until it reaches the ultimate gain, at which the output of the control loop has stable and consistent oscillations.

Generally, the Ziegler–Nichols have to be implemented through trial-and-error process. Therefore, we can break down the overall PID control diagram on this self-balancing robot into three relatively independent loops: vertical control loop, velocity control loop and sheering control loop. Not all of these loops contain all the three PID parameters so the control process could be simplified. The component of these three control loops are defined and tuned with the procedure discussed below.

A. *Vertical Control Loop Tuning*

In order to guarantee the stability of the robot on the ground, the vertical control loop must be tuned to appropriate parameters at the first place. As the robot must be robust and agile enough to endure external interferences, the derivative parameter is added to its vertical control loop. The P and D parameters could be tuned with the method of trial and error through the following steps.

With the MPU6050 capable of censoring the static attitude of our robot, the first step of vertical control loop tuning is simply to find its equilibrium point. When we put our robot upright on the ground, the LCD screen would display the tilt angle of the robot. As a result, the equilibrium point could be found by letting the robot sway from one side to the other before its collapsing down. The equilibrium point is found approximately at the angle of 3 degrees without external interference and it shall be considered as the balance tilt angle in the control loop. So, the tilt angle set point of our controller is 3 degrees.

After determination of balance point, we adopt the PD controller in the car’s vertical control loop. P is the first parameter to be tuned as it determines the force with which the robot will correct itself. A lower P shows robot’s inability to balance itself and a higher P will lead to the violent behavior. As the vertical control loop is designed to be negative feedback, the K_p is therefore a positive value. According to the theoretical knowledge learnt at automotive control courses, the P parameter K_p in the vertical loop would determine the tilt angle where the DC motor would accelerate to its maximum speed. With the analysis of inverted pendulum system before, it is reasonable to set the equilibrium limit at around 25 degrees. As the full duty ratio of our motor encoder PWM is set at 7200, the K_p in correspondence to maximum speed should be set at around 350.

Take K_p initial value to be 350 and adjust K_p so that the robot starts to oscillate (move back and forth) about the balance position. K_p should be large enough for the robot to move but not too large otherwise the movement would not be smooth. The upper limit of K_p is found at around 600 where the robot oscillates violently, revealing its unstable state.

The analysis in the tuning of D parameter K_d adopts a similar method. This parameter should also be a positive value with its function in a negative feedback controller. With the K_p set in the step above, the tuning of K_d is realized through increasing it until the robot would move about its balanced position more gentle, and there shouldn’t be any significant overshoots. With the K_d set at around 0.5, the overshoot oscillation caused by K_p in the first step is eliminated. Increasing K_d to 1, the robot starts to oscillate the high frequency, revealing the high bias of output at this time. So the upper limit of K_d is about 1.

With the P and D parameters set, our robot at this time could stay stable for a few seconds without the ability to cure large tilt angle interference. This large bias would cause the motor accelerating to the maximum speed without limitation.

B. *Velocity Control Loop Tuning*

As is illustrated, the velocity control of self-balancing robot follows the theory of inverted pendulum system. When the system reveals a forward tilt angle, the motor has to accelerate to the same direction to ‘chase’ that tendency. As a result, the positive feedback loop is adopted for the velocity controller. With constant power apply and running state, the PI controller is selected for velocity control loop as the linear velocity controller. The output of PI controller could be transmitted to the motor encoder to produce PWM waves.

In this PI controller, the parameter P determines the force with which the robot will correct itself. The output torque of DC motor would be determined by K_p here. In the meanwhile, parameter I determines the response time of robot for correcting itself. Higher the K_i, Faster it will response. Because velocity control loop must be robust enough to give agile response to large external interference, I is a significant value. In practical applications, these two parameters has the relationship of $K_i = K_p / 200$ .According to the design demand of self-balancing robot, it only requires to stay stable without collapsing. As a result, the velocity set point could be $v_f = 0$ and the PI controller would implement linear control over the robot.

Based on these practical knowledge, the two parameters are tuned simultaneously. Since the velocity control loop has positive feedback, both of these factors are positive values. In the next stage, they are increased until unstable state. When $K_p=110$, $K_i=0.55$, the robot oscillates again with small external interference. So these two parameters shall be tuned to the smaller value around $K_p=90$, $K_i=0.45$.

After implementing $i-1$ loads on the self-balancing system, the test result reveals that no significant change happened to the velocity loop as this loop mainly controls the two DC motor encoders. The system at this time can remain upright with the load on the ground, revealing the primary success of our PID tuning process.

C. Sheering Control Loop Tuning

The last loop to tune is the steering control loop utilized under remote controlled state. After realizing the basic upright endurance of the self-balancing system, the steering control loop is implemented to test whether the system with loads could undertake active tile angle without falling down. If the self-balancing system could timely 'chase' the tilt angle while moving straight forward actively, then the system is definitely robust. Here we utilize the P control to steering loop to avoid interfering with the vertical and velocity control loops.

The K_p parameter in this loop is to realize negative angular feedback. As a result, it should be a positive value during the tuning process. The K_p value is increased until the robot starts to oscillate violently. The final tuned K_p is set to be 45 .

During the loading process, it seems that K_p shall be increased so that the robot can be controlled to move in a straight line after increasing its mass value. As a result, the final when the system is well-tuned with load on it. From the graph below, we can conclude that the robot at this case could endure the load on it without falling down. The sway problem of robot could be reduced gradually with further PID parameter optimization.

Theoretically, ID values depends on the state of the system. External mechanical structure, physical properties, electrical properties, etc. But practically, it also depends on the external conditions like external atmospheric conditions, etc.

IV. Tuning Result and Signal Analysis

Based on the tuning of the three control loops listed above, we could implement these PID parameters into the overall systematic balancing. After realizing the basic balance of the robot on tile floor, we change the plane material on which the robot balance shall be achieved. The material is the yoga mat made from polyvinyl chloride, which has a much higher surface friction coefficient. This larger friction between the surface and robot wheel has a severe influence on the transient performance, including the rise time, overshoot and settling time.

Implementing the tile floor based tuning parameters, we put the self-balancing robot on the yoga mat, and the basic system parameters are shown in the following graph.

Target Loop	Control Parameter	Value
Vertical Contol Loop	Propotional Kp	350
	Derivative Kd	0.5
Velocity Control Loop	Propotional Kp	90
	Integral Ki	0.45
Direction Control Loop	Propotional Kp	42

Fig. 5. System Parameters before Optimization

Here we utilize the upper-monitor for microcontroller to achieve real-time display of the acquired data. The display is achieved through serial port data transmission, where the baud rate is set at 128000 and figure could be watched through the software interface. The main waveform to be displayed is the data from acceleration sensor module. From the waveform we could see the obscure attitude of the robot. The figure of this system state before optimization tuning is shown as below.

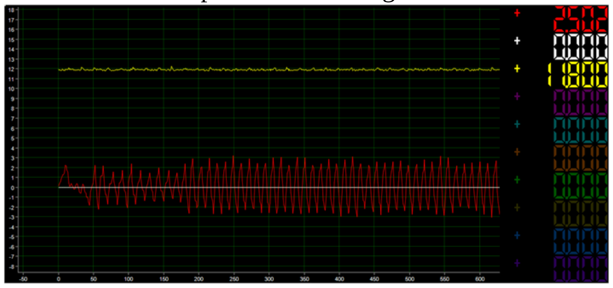

Fig. 6. System Waveform before Optimization

As is shown from the graph, the system performance is comparatively worse after putting on the mat. The small oscillation wave around zero-line reveals that the resetting time is large and steady-state error could not be neglected. With this kind of high-resistance system, the balance could be broken.

Then we test the system by imposing it with heavy load. This kind of exaggeration method is used to further exaggerate the error and bias of the system. The load is selected to be an invariant 1.429 kg dictionary. In order to maintain the extra variants, the dictionary is imposed with the standard that its barycenter must be coincided with the self-balancing system (3 degrees of mechanic balance point).

After implementing this kind of load to the system on mat, the robot car starts to oscillate violently, losing the ability to decrease the oscillation amplitude. From the upper-monitor graph above, the procedure of balance loss could clearly be illustrated.

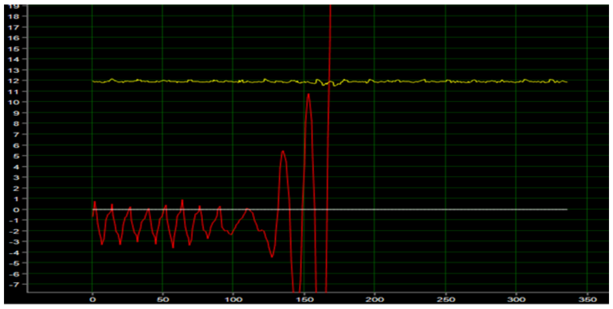

Fig. 7. Control Loss Procedure before Optimization

From Newtonian mechanics' analysis, the reason for losing control of this self-balancing system mainly arises from its increased mass and higher gyrocenter.

The further analysis based on moment equilibrium theory would give out the change tendency after posing heavy load to the robot:

$$I = \sum_i m_i r_i$$

$$M = \sum_i F_i L_i$$

Where m_i is the weight of the part i of 1the balance system, r_i is the distance of each part to the new barycenter of the system. F_i is the external force implemented from each mass part and L_i is the distance between force lines of action and barycenter.

After implementing $i - 1$ loads on the self-balancing system which altogether consists of i parts, the barycenter would raise to a higher height from the robot chassis, leading to less recovery force and more moment for the system. Accordingly, the P parameter would decrease while the D parameter have to be increased.

With higher torque demand and inertia, the system could be optimized through PID parameter tuning. Based on the method of Ziegler–Nichols method, which is demonstrated in part 2, we tuned this system parameters into the following values:

Target Loop	Control Parameter	Value
Vertical Contol Loop	Propotional Kp	300
	Derivative Kd	1
Velocity Control Loop	Propotional Kp	80
	Integral Ki	0.4
Direction Control Loop	Propotional Kp	45

Fig. 8. Tuned Parameters for the system

The tuned system performance also goes through the same testing procedure. The tuned robot is put onto the mat again. Before imposing the load, its waveform is shown as below.

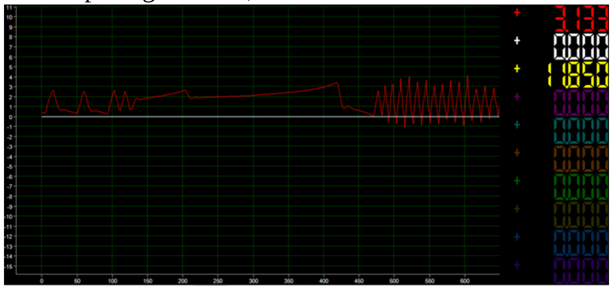

Fig. 9. System Waveform after Optimization

As shown in the graph, the periodic oscillation of the system is decreased in amplitude. The beginning of this kind of oscillation interference is also postponed that the system could maintain perfect balance state on the mat surface for several seconds. Then the load is imposed on to our testing robot again. The experiment on the mat with load illustrates the system state in the following graph.

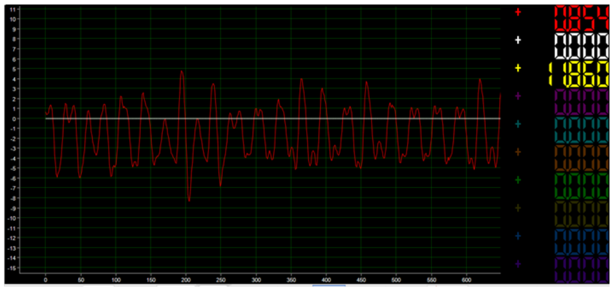

Fig. 10. System Optimization Effect with Load

According to this graph, the system oscillation is limited into a controllable range that could not go beyond control. The further analysis into the waveform could be implemented that this increasing oscillation tendency is weakened periodically, which justifies the better performance of increasing derivative factor value in our PID controller. The oscillation is greatly reduced with increased damping force. The prolonged resetting time also improves the systematic stability.

V. Future Work & Prospects

In this experiment, we got much familiar in the self-balancing control system. While it is true that such system could be tuned finally with PID controller based on the Ziegler–Nichols method, this kind of method cannot achieve the perfect system performance. (As illustrated above, the balance performance for the robot is still not as satisfying as placed on the smooth ground without load.) Such kind of problem could not be resolved by the basic PID controller, which lead us to the future exploration to the modern control theories that are widely adopted in engineering industries. The methods that could be implemented to optimize this system include auto-tuning, which makes the adaptive tuning of devices themselves to achieve intelligent tuning.

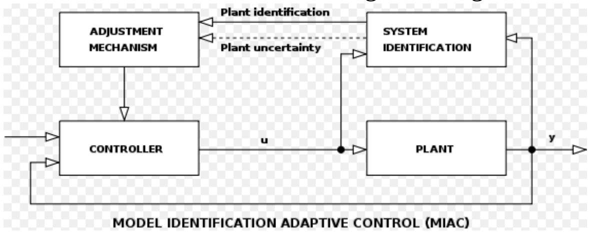

Fig. 11. Adaptive Control & Tuning Scheme

Furthermore, the Ziegler–Nichols method for tuning performs its low efficiency during the test. If it really works, this tuning method is really easy to adopt. But if not, then the seemingly endless trial and error must be adopted. we cannot recommend the Z-N rule or related tuning rules strongly. The idea is straightforward but the application is tricky. So in order to raise the tuning efficiency, this kind of procedure could be finished through the cutting-edged DAP tuning, which realizes parameter tuning on the software platform. This kind of tuning could help us to realize the desired system performance with a short cost of time.

VI. Conclusion

Through this PID control and its parameter tuning process, the self-balancing two-wheel robot could achieve a better balance performance. Alternating the external ground condition as well as imposing heavy loads on the robot exaggerates its control quality for this project. After extracting data from the serial port and watch the real-time attitude of the self-balancing car through the upper-computer interface, we could accelerate our tuning procedure. With signal processing and data mining of these transmitted data, the research into precise control system could be conducted. Advanced controlling system as well as tuning method could be adopted to realize this procedure with higher efficiency and better result.

Acknowledgment

We gratefully acknowledge Dr. Yu Deping for his guidance and provision of the hardware components and would like to thank Teaching Assistant Zhao Yong for his assistance during this experiment and collaboration.

References

[1] Rasoul Sadeghian, Mehdi Tale Masoule, "An Experimental Study on the PID and Fuzzy-PID Controllers on a Designed Two-Wheeled Self Balancing Autonomous Robot (Published Conference Proceedings style)," in *Proc. 4th International Conference on Control, Instrumentation, and Automation*, Qazvin Islamic Azad University, 2016, pp. 313–318.

[2] Hau-Shiue Juang, Kai-Yew Lum, "Design and Control of a Two-Wheel Self-Balancing Robot using the Arduino Microcontroller Board (Published Conference Proceedings style)", in *Proc. 10th IEEE International Conference on Control and Automation,* Hangzhou, 2013, pp. 634–639.

[3] Xia Kun, Wang Hong-Jie, Zhang Zhen-Guo, Chen Jian-Qiang "Design of a Self-Balanced System based on AccuStar Electronic Inclinometer", unpublished.

[4] Barlian Henryranu Prasetio, "Ensemble Kalman Filter and PID Controller
Implementation On Self Balancing Robot. (Published Conference Proceedings style)", in *Proc. 10th International Electronics Symposium (IES),* Hangzhou, 2015, pp. 106–107.

[5] Wu Liu-bo and Zhang Zhen-guo, "Application of inclination sensor in the rail-measuring system," Chinese journal of sensors and actuators, vol. 18, no. 3, pp. 668-670, 2005.

[6] Wang Xiao-ming, Single-chip control for the motor. Beijing University of Aeronautics and Astronautics Press, 2002.

[7] B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[8] J. P. Wilkinson, "Nonlinear resonant circuit devices (Patent style)," U.S. Patent 3 624 12, July 16, 1990.

[9] *IEEE Criteria for Class IE Electric Systems* (Standards style), IEEE Standard 308, 1969.

[10] Qianyun Li, Guowei Gao, Xiuqin Wu and Zhenfeng Li, "The inclinometer research for attitude control," *2008International conference on intelligent computation technology and automation (ICICTA2008),* Changsha(China), Oct. 2008, pp. 512-515.

Author Information

[1] Yin Si-kai

Through this group project on self-balancing system, I successfully utilized my knowledge obtained from the microcontroller and automatic control courses. The lack in STM 32 ARM architecture posed great hinder in our project process in the first stage. With the perseverance and diligence, we finished reading guidebooks, references (which are all English thesis and documents) and the structure of the STM 32 project. The programming as well as tuning of the self-balancing system is a difficult but worthwhile procedure. It greatly tested our learning and innovation ability, which are all crucial elements for future research and exploration.

In regard with the self-balancing system, I have witnessed so many fascinating videos, advertisements and reviews over this kind of flywheel or Segway. This kind of personal transportation vehicle is considered by many to be the promising alternative to the private cars and buses in the streets nowadays. The energy storage and efficiency problem now still hinders their wide application, but they are still valuable in providing a possible way to develop the electrical vehicles of the next generation. This kind of energetic field is promising and worthy of our devotion and exploration.

Finally, I would like show my thanks to my teammates as well as our instructors, without whom this kind of project can hardly be finished on time. Also, I have obtained a lot of practical abilities and deeper understanding in systematic control. I would learn more about this direction which combines the mechanics and electronics together to raise new ideas. Hopefully, they could be utilized someday to make the world a better place.

[2] Zhang Qing-bo

In this project of Two-Wheeled Self-Balancing Robot, my roles are checking the programs of robot and trying to understand the meaning of each codes so that we can write the new PID parameters to the codes exactly when we are trying to test programs. Beside these, I also response for writing the outline of report.

For myself, I think this project is a little challenging. Hard but good for myself to learn new knowledge in which I interest.To understand the control principle of Two-Wheeled Self-Balancing Robot and the mean of PID control, I search for lots of knowledge which I haven't heard about before, which promotes my ability of self-learning. What's more, I must review the knowledge of language C when checking the programs of robot since the original codes are written in C, which I think is a good chance to know how to use C in microcontroller.

It is cooperation in the whole process of this project that impress me most. And I believe without partners' help and their efforts I wouldn't have finished this project in time. Thanks for my partners.
Nowadays Self-Balancing Robot become more and more popular, and I wish to learn more knowledge about it.

[3] Yang Deng-feng

In the last several weeks, I and my teammates did some research and experiments on Ziegler−Nichols PID Tuning based on a Two-Wheeled Self-Balancing Robot，through which I have learnt the working principles of the Self-Balancing Robot and applied the knowledge that I learned in the class to the study on the Self-Balancing Robot.

During the process, I met some problems that included not only program codes but also mathematical，physical and electrical confusion. Though it was the first time that I met such a professional practical problem，I didn't' t give up. Our group discussed these problems and made a division of labor according to our specialties. We began to search information and data from the library and professional websites to find what we need. Finally，our hard work paid off and we have found data we needed and solved those problems we met. When everything was prepared，we successfully finished the study on the Self-Balancing Robot as we planned before. When we got the experiment data and made analysis，we got precious information and data which can help us optimize the Self-Balancing Robot.

Through the study in the whole process, not only did I know how to use what I have learnt to solve practical problems，but also I have developed my abilities to make good cooperation，which taught and influenced me a lot.

Main Program: STM32 Hardware Self-balancing System

```
#include "sys.h"
//设置向量表偏移地址
//NVIC_VectTab:基址
//Offset:偏移量
void MY_NVIC_SetVectorTable(u32 NVIC_VectTab, u32 Offset)
{
    SCB->VTOR = NVIC_VectTab|(Offset & (u32)0x1FFFFF80);//设置NVIC 的向量表偏移寄存器
    //用于标识向量表是在 CODE 区还是在 RAM 区
}
//设置 NVIC 分组
//NVIC_Group:NVIC 分组 0~4 总共 5 组
void MY_NVIC_PriorityGroupConfig(u8 NVIC_Group)
{
    u32 temp,temp1;
    temp1=(~NVIC_Group)&0x07;//取后三位
    temp1<<=8;
    temp=SCB->AIRCR;  //读取先前的设置
    temp&=0X0000F8FF; //清空先前分组
    temp|=0X05FA0000; //写入钥匙
    temp|=temp1;
    SCB->AIRCR=temp;  //设置分组
}
//设置 NVIC
//NVIC_PreemptionPriority:抢占优先级
//NVIC_SubPriority       :响应优先级
//NVIC_Channel           :中断编号
//NVIC_Group             :中断分组 0~4
//注意优先级不能超过设定的组的范围!否则会有意想不到的错误
//组划分:
//组 0:0 位抢占优先级,4 位响应优先级
//组 1:1 位抢占优先级,3 位响应优先级
//组 2:2 位抢占优先级,2 位响应优先级
//组 3:3 位抢占优先级,1 位响应优先级
//组 4:4 位抢占优先级,0 位响应优先级
//NVIC_SubPriority 和 NVIC_PreemptionPriority 的原则是,数值越小,越优先
void MY_NVIC_Init(u8 NVIC_PreemptionPriority,u8 NVIC_SubPriority,u8 NVIC_Channel,u8 NVIC_Group)
{
    u32 temp;
    MY_NVIC_PriorityGroupConfig(NVIC_Group);//设置分组
    temp=NVIC_PreemptionPriority<<(4-NVIC_Group);
    temp|=NVIC_SubPriority&(0x0f>>NVIC_Group);
    temp&=0xf;//取低四位
    NVIC->ISER[NVIC_Channel/32]|=(1<<NVIC_Channel%32);//使能中断位(要清除的话,相反操作就 OK)
    NVIC->IP[NVIC_Channel]|=temp<<4;//设置响应优先级和抢断优先级

}
//外部中断配置函数
//只针对 GPIOA~G;不包括 PVD,RTC 和 USB 唤醒这三个
//参数:
//GPIOx:0~6,代表 GPIOA~G
//BITx:需要使能的位;
//TRIM:触发模式,1,下升沿;2,上降沿;3，任意电平触发
//该函数一次只能配置 1 个 IO 口,多个 IO 口,需多次调用
//该函数会自动开启对应中断,以及屏蔽线
void Ex_NVIC_Config(u8 GPIOx,u8 BITx,u8 TRIM)
{
    u8 EXTADDR;
    u8 EXTOFFSET;
    EXTADDR=BITx/4;//得到中断寄存器组的编号
    EXTOFFSET=(BITx%4)*4;
    RCC->APB2ENR|=0x01;//使能 io 复用时钟
    AFIO->EXTICR[EXTADDR]&=~(0x000F<<EXTOFFSET);//清除原来设置！！！
    AFIO->EXTICR[EXTADDR]|=GPIOx<<EXTOFFSET;//EXTI.BITx 映射到 GPIOx.BITx
    //自动设置
    EXTI->IMR|=1<<BITx;//  开启 line BITx 上的中断
   //EXTI->EMR|=1<<BITx;//不屏蔽 line BITx 上的事件 (如果不屏蔽这句,在硬件上是可以的,但是在软件仿真的时候无法进入中断!)
    if(TRIM&0x01)EXTI->FTSR|=1<<BITx;//line BITx 上事件下降沿触发
    if(TRIM&0x02)EXTI->RTSR|=1<<BITx;//line BITx 上事件上升降沿触发
}
//不能在这里执行所有外设复位!否则至少引起串口不工作.
//把所有时钟寄存器复位
void MYRCC_DeInit(void)
{
    RCC->APB1RSTR = 0x00000000;//复位结束
    RCC->APB2RSTR = 0x00000000;

    RCC->AHBENR = 0x00000014;  //睡眠模式闪存和 SRAM 时钟使能.其他关闭.
    RCC->APB2ENR = 0x00000000; //外设时钟关闭.
    RCC->APB1ENR = 0x00000000;
    RCC->CR |= 0x00000001;     //使能内部高速时钟 HSION

    RCC->CFGR &= 0xF8FF0000;   //复位SW[1:0],HPRE[3:0],PPRE1[2:0],PPRE2[2:0],ADCPRE[1:0],MCO[2:0]

    RCC->CR &= 0xFEF6FFFF;     //复位 HSEON,CSSON,PLLON
    RCC->CR &= 0xFFFBFFFF;     //复位 HSEBYP
    RCC->CFGR &= 0xFF80FFFF;   //复位 PLLSRC, PLLXTPRE, PLLMUL[3:0] and USBPRE
    RCC->CIR = 0x00000000;     //关闭所有中断
    //配置向量表
#ifdef  VECT_TAB_RAM
    MY_NVIC_SetVectorTable(0x20000000, 0x0);
#else
    MY_NVIC_SetVectorTable(0x08000000,0x0);
#endif
}
//THUMB 指令不支持汇编内联
//采用如下方法实现执行汇编指令 WFI
__asm void WFI_SET(void)
{
    WFI;
}
//关闭所有中断
__asm void INTX_DISABLE(void)
{
    CPSID I;
}
//开启所有中断
__asm void INTX_ENABLE(void)
{
    CPSIE I;
}
//设置栈顶地址
//addr:栈顶地址
__asm void MSR_MSP(u32 addr)
{
      MSR MSP, r0            //set Main Stack value
      BX r14
}

//进入待机模式
void Sys_Standby(void)
{
    SCB->SCR|=1<<2;//使能 SLEEPDEEP 位 (SYS->CTRL)
    RCC->APB1ENR|=1<<28;        //使能电源时钟
    PWR->CSR|=1<<8;             //设置 WKUP 用于唤醒
    PWR->CR|=1<<2;              //清除 Wake-up 标志
    PWR->CR|=1<<1;              //PDDS 置位
    WFI_SET();            //执行 WFI 指令
}
//系统软复位
void Sys_Soft_Reset(void)
{
    SCB->AIRCR =0X05FA0000|(u32)0x04;
}
//JTAG 模式设置,用于设置 JTAG 的模式
//mode:jtag,swd 模式设置;00,全使能;01,使能 SWD;10,全关闭;
//#define JTAG_SWD_DISABLE     0X02
//#define SWD_ENABLE           0X01
//#define JTAG_SWD_ENABLE      0X00
void JTAG_Set(u8 mode)
{
    u32 temp;
    temp=mode;
    temp<<=25;
    RCC->APB2ENR|=1<<0;        //开启辅助时钟
    AFIO->MAPR&=0XF8FFFFFF; //清除 MAPR 的[26:24]
    AFIO->MAPR|=temp;          //设置 jtag 模式
}
//系统时钟初始化函数
//pll:选择的倍频数，从 2 开始，最大值为 16
void Stm32_Clock_Init(u8 PLL)
{
    unsigned char temp=0;
    MYRCC_DeInit();          //复位并配置向量表
    RCC->CR|=0x00010000;    //外部高速时钟使能 HSEON
    while(!(RCC->CR>>17));//等待外部时钟就绪
    RCC->CFGR=0X00000400; //APB1=DIV2;APB2=DIV1;AHB=DIV1;
    PLL-=2;//抵消 2 个单位
    RCC->CFGR|=PLL<<18;     //设置 PLL 值 2~16
    RCC->CFGR|=1<<16;      //PLLSRC ON
    FLASH->ACR|=0x32;      //FLASH 2 个延时周期

    RCC->CR|=0x01000000;    //PLLON
    while(!(RCC->CR>>25));//等待 PLL 锁定
    RCC->CFGR|=0x00000002;//PLL 作为系统时钟
    while(temp!=0x02)       //等待 PLL 作为系统时钟设置成功
    {
        temp=RCC->CFGR>>2;
        temp&=0x03;
    }
}
```

Main Program: Initialization

```
#include "sys.h"
u8 Way_Angle=2;                                  //获取角度的算法，1：四元数  2：卡尔曼  3：互补滤波
u8 Flag_Qian,Flag_Hou,Flag_Left,Flag_Right,Flag_sudu=2; //蓝牙遥控相关的变量
u8  Flag_Stop=1,Flag_Show=0;                     //停止标志位和 显示标志位 默认停止 显示打开
int Encoder_Left,Encoder_Right;                  //左右编码器的脉冲计数
int  Moto1,Moto2;                                //电机 PWM 变量 应是 Motor 的
int Temperature;                                 //显示温度
int  Voltage;                                    //电池电压采样相关的变量
float  Angle_Balance,Gyro_Balance,Gyro_Turn; //平衡倾角 平衡陀螺仪 转向陀螺仪
float Show_Data_Mb;                              //全局显示变量，用于显示需要查看的数据
u32 Distance;                                    //超声波测距
u8 delay_50,delay_flag,Bi_zhang=0;               //默认情况下，不开启避障功能
float Acceleration_Z;                            //Z 轴加速度计
int main(void)
{
    Stm32_Clock_Init(9);                 //=====系统时钟设置
    delay_init(72);                      //=====延时初始化
    JTAG_Set(JTAG_SWD_DISABLE);          //=====关闭 JTAG 接口
    JTAG_Set(SWD_ENABLE);                //=====打开 SWD 接口
    LED_Init();                          //=====初始化与 LED
    KEY_Init();                          //=====按键初始化
    OLED_Init();                         //=====OLED 初始化
    uart_init(72,128000);                //=====初始化串口 1
   uart3_init(36,9600);                  //=====串口 3 初始化
   MiniBalance_PWM_Init(7199,0);         //=====初始化 PWM 10KHZ，用于驱动电机
    Encoder_Init_TIM2();                 //=====编码器接口
    Encoder_Init_TIM4();                 //=====初始化编码器 2
    Adc_Init();                          //=====adc 初始化
    IIC_Init();                          //=====模拟 IIC 初始化
   MPU6050_initialize();                 //=====MPU6050 初始化
   DMP_Init();                           //=====初始化 DMP
    TIM3_Cap_Init(0XFFFF,72-1);          //=====超声波初始化
    EXTI_Init();                         //=====MPU6050 5ms 定时中断初始化
    while(1)
       {
```

```
                if(Flag_Show==0)                //使用 APP 和 OLED 显示屏
                {
                        APP_Show();
                        oled_show();            //===显示屏打开
                }
                else                            //使用上位机，关闭 app 监控
部分和 OLED 显示屏
                {
                      DataScope();              //开启上位机
                }
               delay_flag=1;
                delay_50=0;
                while(delay_flag);        //通过 MPU6050 的 INT 中断实现的
50ms 精准延时
        }
}
```

Main Program: Attitude Acquisition

```
#ifndef __FILTER_H
#define __FILTER_H

extern float angle, angle_dot;
void Kalman_Filter(float Accel,float Gyro);
void Yijielvbo(float angle_m, float gyro_m);
#endif

/*****************************************************************
***********
信号处理滤波算法实现函数，分别为卡尔曼和互补滤波
*****************************************************************
***********/
float K1 =0.02;
float angle, angle_dot;
float Q_angle=0.001;// 过程噪声的协方差
float  Q_gyro=0.003;//0.003 过程噪声的协方差 过程噪声的协方差为一个
一行两列矩阵
float R_angle=0.5;// 测量噪声的协方差 既测量偏差
float dt=0.005;//
char    C_0 = 1;
float Q_bias, Angle_err;
float PCt_0, PCt_1, E;
float K_0, K_1, t_0, t_1;
float Pdot[4] ={0,0,0,0};
float PP[2][2] = { { 1, 0 },{ 0, 1 } };

/*****************************************************************
***********
函数功能：简易卡尔曼滤波
入口参数：加速度、角速度
返回  值：无
*****************************************************************
***********/
void Kalman_Filter(float Accel,float Gyro)
{
    angle+=(Gyro - Q_bias) * dt; //先验估计
    Pdot[0]=Q_angle - PP[0][1] - PP[1][0]; // Pk-先验估计误差协方差的微
分

    Pdot[1]=-PP[1][1];
    Pdot[2]=-PP[1][1];
    Pdot[3]=Q_gyro;
    PP[0][0] += Pdot[0] * dt;    // Pk-先验估计误差协方差微分的积分
    PP[0][1] += Pdot[1] * dt;    // =先验估计误差协方差
    PP[1][0] += Pdot[2] * dt;
    PP[1][1] += Pdot[3] * dt;

    Angle_err = Accel - angle;  //zk-先验估计

    PCt_0 = C_0 * PP[0][0];
    PCt_1 = C_0 * PP[1][0];

    E = R_angle + C_0 * PCt_0;

    K_0 = PCt_0 / E;
    K_1 = PCt_1 / E;

    t_0 = PCt_0;
    t_1 = C_0 * PP[0][1];

    PP[0][0] -= K_0 * t_0;        //后验估计误差协方差
    PP[0][1] -= K_0 * t_1;
    PP[1][0] -= K_1 * t_0;
    PP[1][1] -= K_1 * t_1;

    angle  += K_0 * Angle_err;   //后验估计
    Q_bias    += K_1 * Angle_err;   //后验估计
    angle_dot     = Gyro - Q_bias;  //输出值(后验估计)的微分=角速度
}
/*****************************************************************
***********
函数功能：一阶互补滤波
入口参数：加速度、角速度
返回  值：无
*****************************************************************
***********/
void Yijielvbo(float angle_m, float gyro_m)
{
     angle = K1 * angle_m+ (1-K1) * (angle + gyro_m * 0.005);
}
```

Main Program: Control Loop

```
#ifndef __CONTROL_H
#define __CONTROL_H
#include "sys.h"
#define PI 3.14159265
#define ZHONGZHI 3
#define DIFFERENCE 100
externint Balance_Pwm,Velocity_Pwm,Turn_Pwm;
int EXTI15_10_IRQHandler(void);
int balance(float angle,float gyro);
int velocity(int encoder_left,int encoder_right);
int turn(int encoder_left,int encoder_right,float gyro);
void Set_Pwm(int moto1,int moto2);
void Key(void);
void Xianfu_Pwm(void);
u8 Turn_Off(float angle, int voltage);
void Get_Angle(u8 way);
int myabs(int a);
int Pick_Up(float Acceleration,float Angle,int encoder_left,int encoder_right);
int Put_Down(float Angle,int encoder_left,int encoder_right);
#endif

#include "filter.h"
int Balance_Pwm,Velocity_Pwm,Turn_Pwm;
u8 Flag_Target;
/*****************************************************************
***********
函数功能：主控制环函数
```

```
            5ms 定时中断由 MPU6050 的 INT 引脚触发
            严格保证采样和数据处理的时间同步
******************************************************************
***********/
int EXTI9_5_IRQHandler(void)
{
     if(PBin(5)==0)
    {
          EXTI->PR=1<<5;
//清除 LINE5 上的中断标志位
           Flag_Target=!Flag_Target;
          if(delay_flag==1)
            {
               if(++delay_50==10)  delay_50=0,delay_flag=0;          //给主函
数提供 50ms 的精准延时
            }
          if(Flag_Target==1)
//5ms 读取一次陀螺仪和加速度计的值，更高的采样频率可以改善卡尔曼
滤波和互补滤波的效果
           {
           Get_Angle(Way_Angle);
//===更新姿态
           return 0;
           }
//10ms 控制一次，为了保证 M 法测速的时间基准，首先读取编码器数据
           Encoder_Left=-Read_Encoder(2);
//===读取编码器的值，因为两个电机的旋转了 180 度的，所以对其中一
个取反，保证输出极性一致
           Encoder_Right=Read_Encoder(4);
//===读取编码器的值
       Get_Angle(Way_Angle);
//===更新姿态
           Read_Distane();
//===获取超声波测量距离值
       if(Bi_zhang==0)Led_Flash(100);
//===LED 闪烁;常规模式  1s 改变一次指示灯的状态
           if(Bi_zhang==1)Led_Flash(0);
//===LED 闪烁;避障模式  指示灯常亮
       Voltage=Get_battery_volt();
//===获取电池电压
           Key();
//===扫描按键状态  单击双击可以改变小车运行状态
           Balance_Pwm                 =balance(Angle_Balance,Gyro_Balance);
//===平衡 PID 控制
         Velocity_Pwm=velocity(Encoder_Left,Encoder_Right);
//===速度环 PID 控制     记住，速度反馈是正反馈，就是小车快的时候要
慢下来就需要再跑快一点
      Turn_Pwm                 =turn(Encoder_Left,Encoder_Right,Gyro_Turn);
//===转向环 PID 控制
          Moto1=Balance_Pwm-Velocity_Pwm+Turn_Pwm;
//===计算左轮电机最终 PWM
       Moto2=Balance_Pwm-Velocity_Pwm-Turn_Pwm;
//===计算右轮电机最终 PWM
           Xianfu_Pwm();
//===PWM 限幅

    if(Pick_Up(Acceleration_Z,Angle_Balance,Encoder_Left,Encoder_Right))
//===检查是否小车被那起
           Flag_Stop=1;
//===如果被拿起就关闭电机
            if(Put_Down(Angle_Balance,Encoder_Left,Encoder_Right))
//===检查是否小车被放下
            Flag_Stop=0;
//===如果被放下就启动电机
          if(Turn_Off(Angle_Balance,Voltage)==0)
//===如果不存在异常
            Set_Pwm(Moto1,Moto2);
//===赋值给 PWM 寄存器
    }
     return 0;
}

/******************************************************************
***********
函数功能：直立 PD 控制
入口参数：角度、角速度
返回  值：直立控制 PWM
******************************************************************
***********/
int balance(float Angle,float Gyro)
{
    float Bias,kp=300,kd=1;
     int balance;
     Bias=Angle+3;                        //===求出平衡的角度中值  和机械相
关
     balance=kp*Bias+Gyro*kd;     //===计算平衡控制的电机  PWM   PD
控制    kp 是 P 系数  kd 是 D 系数
     return balance;
}

/******************************************************************
***********
函数功能：速度 PI 控制
入口参数：左轮编码器、右轮编码器
返回  值：速度控制 PWM
******************************************************************
***********/
int velocity(int encoder_left,int encoder_right)
{
      static float Velocity,Encoder_Least,Encoder,Movement;
     static float Encoder_Integral,Target_Velocity;
     float kp=80,ki=0.4;
     //=============    遥    控    前    进    后    退    部    分
======================//
     if(Bi_zhang==1&&Flag_sudu==1)                    Target_Velocity=45;
//如果进入避障模式,自动进入低速模式
    else                                        Target_Velocity=45;
      if(1==Flag_Qian)          Movement=Target_Velocity/Flag_sudu;
//===前进标志位置 1
      else if(1==Flag_Hou) Movement=-Target_Velocity/Flag_sudu;
//===后退标志位置 1
     else    Movement=0;
     if(Bi_zhang==1&&Distance<500&&Flag_Left!=1&&Flag_Right!=1)
//避障标志位置 1 且非遥控转弯的时候，进入避障模式
     Movement=-Target_Velocity/Flag_sudu;
   //=============速度 PI 控制器=======================//
      Encoder_Least                    =(Encoder_Left+Encoder_Right)-0;
//===获取最新速度偏差==测量速度（左右编码器之和）-目标速度（此处
为零）
      Encoder *= 0.8;
//===一阶低通滤波器
      Encoder += Encoder_Least*0.2;
//===一阶低通滤波器
```

```
        Encoder_Integral                                  +=Encoder;
//===积分出位移 积分时间：10ms
        Encoder_Integral=Encoder_Integral-Movement;
//===接收遥控器数据，控制前进后退
        if(Encoder_Integral>10000)    Encoder_Integral=10000;
//===积分限幅
        if(Encoder_Integral<-10000)   Encoder_Integral=-10000;
//===积分限幅
        Velocity=Encoder*kp+Encoder_Integral*ki;
//===速度控制
        if(Turn_Off(Angle_Balance,Voltage)==1||Flag_Stop==1)
Encoder_Integral=0;        //===电机关闭后清除积分
       return Velocity;
}
```

/***
函数功能：转向控制
入口参数：左轮编码器、右轮编码器、Z 轴陀螺仪
返回　值：转向控制 PWM
***/

```
int turn(int encoder_left,int encoder_right,float gyro)//转向控制
{
      static                                                        float
Turn_Target,Turn,Encoder_temp,Turn_Convert=0.9,Turn_Count;
       float Turn_Amplitude=88/Flag_sudu,Kp=45,Kd=0;
       //=============遥控左右旋转部分，要实现旋转则需要 PD 控制
======================//
    if(1==Flag_Left||1==Flag_Right)                        //这一部分主
要是根据旋转前的速度调整速度的起始速度，增加小车的适应性
        {
            if(++Turn_Count==1)
            Encoder_temp=myabs(encoder_left+encoder_right);
            Turn_Convert=50/Encoder_temp;
            if(Turn_Convert<0.6)Turn_Convert=0.6;
            if(Turn_Convert>3)Turn_Convert=3;
        }
       else
        {
            Turn_Convert=0.9;
            Turn_Count=0;
            Encoder_temp=0;
        }
        if(1==Flag_Left)                   Turn_Target-=Turn_Convert;
        else if(1==Flag_Right)           Turn_Target+=Turn_Convert;
        else Turn_Target=0;
      if(Turn_Target>Turn_Amplitude)          Turn_Target=Turn_Amplitude;
//===转向速度限幅
       if(Turn_Target<-Turn_Amplitude) Turn_Target=-Turn_Amplitude;
        if(Flag_Qian==1||Flag_Hou==1)   Kd=0.5;
        else  Kd=0;                                              //转向的时候
取消陀螺仪的纠正
    //=============转向 PD 控制器======================//
        Turn=-Turn_Target*Kp  -gyro*Kd;                  //===结合 Z 轴
陀螺仪进行 PD 控制
       return Turn;
}
```

/***
函数功能：赋值给 PWM 寄存器
入口参数：左轮 PWM、右轮 PWM
返回　值：无
***/

```
void Set_Pwm(int moto1,int moto2)
{
        if(moto1<0)         AIN2=1,          AIN1=0;
            else                AIN2=0,          AIN1=1;
            PWMA=myabs(moto1);
           if(moto2<0)  BIN1=0,         BIN2=1;
            else              BIN1=1,            BIN2=0;
            PWMB=myabs(moto2);
}
```

/***
函数功能：限制 PWM 赋值
入口参数：无
返回　值：无
***/

```
void Xianfu_Pwm(void)
{
       int  Amplitude=6900;                             //===PWM 满幅是 7200
限制在 6900 范围内防止电机过速
       if(Flag_Qian==1)    Moto1+=DIFFERENCE;    //DIFFERENCE 定义为
小车二轮同步修正量，确保二轮不存在差异
       if(Flag_Hou==1)     Moto2-=DIFFERENCE;
      if(Moto1<-Amplitude) Moto1=-Amplitude;
       if(Moto1>Amplitude)    Moto1=Amplitude;    //左轮限幅
       if(Moto2<-Amplitude) Moto2=-Amplitude;
       if(Moto2>Amplitude)    Moto2=Amplitude;    //右轮限幅

}
```

/***
函数功能：按键修改小车运行状态
入口参数：无
返回　值：无
***/

```
void Key(void)
{
    u8 tmp,tmp2;
    tmp=click_N_Double(50);
    if(tmp==1)Flag_Stop=!Flag_Stop;//单击控制小车的启停
    if(tmp==2)Flag_Show=!Flag_Show;//双击控制小车的显示状态
    tmp2=Long_Press();
   if(tmp2==1) Bi_zhang=!Bi_zhang;         //长按控制小车是否进入超声波
避障模式
}
```

/***
函数功能：异常关闭电机
入口参数：倾角和电压
返回　值：1：异常　0：正常
***/

```
u8 Turn_Off(float angle, int voltage)
{
          u8 temp;
            if(angle<-40||angle>40||1==Flag_Stop||voltage<1110)//电池电压低
于 11.1V 关闭电机
            {
//===倾角大于 40 度关闭电机
```

```
        temp=1;
//===Flag_Stop 置 1 关闭电机
          AIN1=0;
          AIN2=0;
          BIN1=0;
          BIN2=0;
        }
          else
        temp=0;
        return temp;
}

/*****************************************************************
***********
函数功能：获取角度
入口参数：获取角度的算法 1：DMP   2：卡尔曼 3：互补滤波
返回  值：无
*****************************************************************
***********/
void Get_Angle(u8 way)
{
         float Accel_Y,Accel_X,Accel_Z,Gyro_Y,Gyro_Z;
          Temperature=Read_Temperature();      //===读取 MPU6050 内
置温度传感器数据，近似表示主板温度。
         if(way==1)                            //===DMP 的读取在数
据采集中断提醒的时候，严格遵循时序要求
          {
                 Read_DMP();                          //===读取加速度、
角速度、倾角
                 Angle_Balance=Pitch;              //===更新平衡倾角
                 Gyro_Balance=gyro[1];             //===更新平衡角速度
                 Gyro_Turn=gyro[2];                 //===更新转向角速度
                Acceleration_Z=accel[2];        //===更新 Z 轴加速度计
          }
        else
        {

    Gyro_Y=(I2C_ReadOneByte(devAddr,MPU6050_RA_GYRO_YOUT_H)
<<8)+I2C_ReadOneByte(devAddr,MPU6050_RA_GYRO_YOUT_L);      //
读取 Y 轴陀螺仪

    Gyro_Z=(I2C_ReadOneByte(devAddr,MPU6050_RA_GYRO_ZOUT_H)
<<8)+I2C_ReadOneByte(devAddr,MPU6050_RA_GYRO_ZOUT_L);      //
读取 Z 轴陀螺仪

Accel_X=(I2C_ReadOneByte(devAddr,MPU6050_RA_ACCEL_XOUT_H)<
<8)+I2C_ReadOneByte(devAddr,MPU6050_RA_ACCEL_XOUT_L); //读取
X 轴加速度计

    Accel_Z=(I2C_ReadOneByte(devAddr,MPU6050_RA_ACCEL_ZOUT_
H)<<8)+I2C_ReadOneByte(devAddr,MPU6050_RA_ACCEL_ZOUT_L);  //
读取 Z 轴加速度计
         if(Gyro_Y>32768)   Gyro_Y-=65536;                    //
数据类型转换   也可通过 short 强制类型转换
          if(Gyro_Z>32768)   Gyro_Z-=65536;                   //
数据类型转换
      if(Accel_X>32768) Accel_X-=65536;                  //数据
类型转换
         if(Accel_Z>32768)  Accel_Z-=65536;                  //数
据类型转换
          Gyro_Balance=-Gyro_Y;                              //
更新平衡角速度
           Accel_Y=atan2(Accel_X,Accel_Z)*180/PI;                 //计
算倾角
          Gyro_Y=Gyro_Y/16.4;                                    //
陀螺仪量程转换
        if(Way_Angle==2)         Kalman_Filter(Accel_Y,-Gyro_Y);// 卡 尔
曼滤波
           else if(Way_Angle==3)     Yijielvbo(Accel_Y,-Gyro_Y);      //互补
滤波
         Angle_Balance=angle;                                    //更
新平衡倾角
           Gyro_Turn=Gyro_Z;
//更新转向角速度
           Acceleration_Z=Accel_Z;
//===更新 Z 轴加速度计
        }
}
/*****************************************************************
***********
函数功能：绝对值函数
入口参数：int
返回  值：unsigned int
*****************************************************************
***********/
int myabs(int a)
{
       int temp;
        if(a<0)   temp=-a;
       else temp=a;
       return temp;
}
/*****************************************************************
***********
函数功能：检测小车是否被拿起（通用算法）
入口参数：int
返回  值：unsigned int
*****************************************************************
***********/
int Pick_Up(float Acceleration,float Angle,int encoder_left,int encoder_right)
{
     static u16 flag,count0,count1,count2;
    if(flag==0)
//第一步
     {
          if(myabs(encoder_left)+myabs(encoder_right)<30)
//条件 1，小车接近静止
            count0++;
          else
          count0=0;
          if(count0>10)
            flag=1,count0=0;
     }
     if(flag==1)
//进入第二步
     {
           if(++count1>200)                                   count1=0,flag=0;
//超时不再等待 2000ms

if(Acceleration>26000&&(Angle>(-20+ZHONGZHI))&&(Angle<(20+ZHO
NGZHI)))     //条件 2，小车是在 0 度附近被拿起
           flag=2;
     }
     if(flag==2)
//第三步
```

```
      {
            if(++count2>100)                                              count2=0,flag=0;
//超时不再等待 1000ms
           if(myabs(encoder_left+encoder_right)>135)
//条件 3，小车的轮胎因为正反馈达到最大的转速
          {
                flag=0;
                return                                                                  1;
//检测到小车被拿起
             }
      }
    return 0;
}
/****************************************************************
***********
函数功能：检测小车是否被放下
入口参数：int
返回  值：unsigned int
****************************************************************
***********/
int Put_Down(float Angle,int encoder_left,int encoder_right)
{
      static u16 flag,count;
      if(Flag_Stop==0)                                  //防止误检
     return 0;
      if(flag==0)
      {

if(Angle>(-10+ZHONGZHI)&&Angle<(10+ZHONGZHI)&&encoder_left==
0&&encoder_right==0)              //条件 1，小车是在 0 度附近的
               flag=1;
      }
      if(flag==1)
      {
            if(++count>50)                                                              //
超时不再等待  500ms
            {
                count=0;flag=0;
            }

if(encoder_left>3&&encoder_right>3&&encoder_left<60&&encoder_right<
60)                          //条件 2，小车的轮胎在未上电的时候被人为转动
          {
                flag=0;
                flag=0;
                return  1;                                                              //
检测到小车被放下
             }
      }
    return 0;
}
```