VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

PROJECT REPORT

EEE-4020 EMBEDDED SYSTEM DESIGN

PROJECT NAME:

SELF BALANCING ROBOT USING PID CONTROLLER IMPLEMENTED ON STM32F407

Team Members:

1. Prem Sharan
2. Ketaka Singh

Submitted To:

Prof. SELVAKUMAR K

Contents

Abstract:

Recently quite a few work has been finished in the self balancing of gadgets. The idea of self balancing started with the balancing of inverted pendulum. This concept prolonged to design of aircrafts as well. in this venture, we have designed a small model of self balancing robotic using the PID(Proportional, quintessential, spinoff) set of rules. when you consider that then, this technique is the new face of the economic technique manipulate systems. This record reviews the methods worried in self balancing of items. This task became performed as a semester challenge to recognize the correlation of PID on the efficiency of various commercial procedures. right here we focus handiest at presenting a short evaluation at the effectiveness and application of the PID manipulate. This paper has been developed through offering a short advent to manipulate structures and associated terminologies, with addition to the motivations for the assignment. Experimentation and observations were taken, deserves and demerits described with ending on the future enhancements. A version of self balancing robot became developed to recognize the effectiveness of PID in the global of manipulate gadget. Going thru some rigorous checks and experiments, the merits and demerits of the PID control device had been determined. It changed into observed that in spite of many advantages of PID manipulate over beyond methods, nonetheless this device requires quite a few upgrades. it's miles was hoping that the reader gets a very good expertise of the importance of self balancing, the effectiveness and deficiencies of PID manipulate

Literature Review:

[1] Wu Junfeng , Zhang Wanying, Research on Control Method of Two-wheeled Self-balancing Robot
primarily based on Newton dynamics mechanics idea make a study of -wheeled self-balancing robot, a detailed mathematical version of the modeling procedure is furnished, after which, the use of the reasonable approach, linear country-area equations is built up. After that, the LQR controller and state comments controller based on pole placement concept are each designed. After a number of simulation experiments, we get the pleasant closed-loop poles and Q, R matrix, both of which have right simulation curves on the identical disturbance pressure. The results of experiments show that both of them have properly dynamic overall performance and robustness, which additionally proves the machine modeling and controller design are affordable and powerful through these methods. The curves from the LQR controller have a higher dynamic performance as compared with the pole placement country-feedback controller.

[2] Ji-Hyun Park, Baek-Kyu Cho Development of a self-balancing robot with a control moment gyroscope

This paper introduces a -two wheeled self-balancing mobile robotic primarily based on a manipulate moment gyroscope module. two-wheeled cell robots are capable of acquire better mobility and rotation in small spaces and to transport faster than legged robots together with humanoid type robots. because of this, the 2-wheeled cell robot is generally used as a cellular robotic platform. however, to hold its balance, the two-wheeled robotic needs to use actions of its two wheels. while an unexpected disturbance impacts the robot, the robotic keeps its balance with movements of the wheels and tilting of the body. If the disturbance exceeds the response functionality of the robotic, the robot will lose its stability. on the equal time, the safety of the robot may be placed at chance through actions to keep balance. To address those problems, a robotic changed into designed with a manipulate second gyroscope module to enhance stability whilst minimizing motion. whilst a disturbance is implemented to the robotic, the disturbance is estimated through a disturbance observer and the manipulate second gyroscope controller compensates the disturbance. the use of the manipulate second gyroscope module, the robotic can maintain balance with just small moves of its wheels. advanced overall performance and balance have been confirmed with experiments and simulations.

[3] HELLMAN, HANNA SUNNERMAN, HENRIK, Design and control based on the concept of an inverted pendulum

In this thesis a two wheeled self-balancing robot has been designed. those varieties of robots may be primarily based on the bodily problem of an inverted pendulum . The machine in itself calls for lively control which will be solid. the usage of open source microcontroller Arduino Uno and reliable angular and positional statistics the system can be made stable by way of enforcing a controller. A present day and green controller is the LQR - Linear Quadratic Regulator. Being a kingdom space feedback controller the version needs to be an excellent illustration of truth for the reason that output sign depends on the version. in this thesis, the validation method was completed the usage of a PID-regulator. The effects confirmed that the version isn't yet dependable. The reasons for this are mentioned and pointers for future improvement are indexed.

Introduction:

With the arrival of computer systems and the industrialization of approaches, all through man‘s records, there has continually been research to expand ways to refine strategies and extra importantly, to manipulate them using machines autonomously. The cause being to lessen man‘s involvement in those techniques, thereby decreasing the mistake in those strategies. consequently, the sphere of ”manipulate system Engineering” become evolved. manipulate machine Engineering may be defined as the usage of various techniques to govern the operating of a process or upkeep of a constant and preferred environment, be it guide or automatic.

A easy example could be of controlling the temperature in a room. guide manipulate approach the presence of someone at a website who exams the prevailing situations (sensor), compares it with the favored cost (processing) and takes appropriate movement to gain the desired cost (actuator). The trouble with this technique is that it isn't always very dependable as someone is liable to blunders or negligence in his paintings. additionally, every other hassle is that the rate of the system initiated by way of the actuator isn't usually uniform, that means sometimes it may occur quicker than required or now and again it may be sluggish. the solution of this hassle turned into to apply a micro-controller to control the system. The micro-controller is programmed to control the method, in step with given specifications, related in a circuit , fed the favoured conditions and thereby controls the process to hold the favored condition. The advantage of this process is that no human intervention is needed on this process. also, the charge of the procedure is uniform

fundamental manipulate machine. The microcontroller is on the heart of any control machine. it's for a completely critical aspect therefore, its desire of choice must be made carefully based totally on the requirements of the device. The micro-controller gets an input from the user. This enter defines the preferred condition of the gadget. The micro-controller also gets a comments enter from the sensor. This sensor is hooked up to the output of the system, the statistics of that is fed back to the input. The microprocessor, based on its programming, performs diverse calculations and gives an output to the actuator. The actuator, based totally at the output, controls the plant to try to preserve the ones situations. An example can be a motor driver driving a motor wherein the motor driver is the actuator and the motor is the plant. The motor, accordingly rotates at a given velocity. The sensor connected reads the circumstance of the plant at the existing time and feeds it again to the micro-controller. The micro-controller again compares, makes calculations and consequently, the cycle repeats itself. This process is repetitive and endless whereby the micro-controller continues the desired conditions

METHODOLOGY:

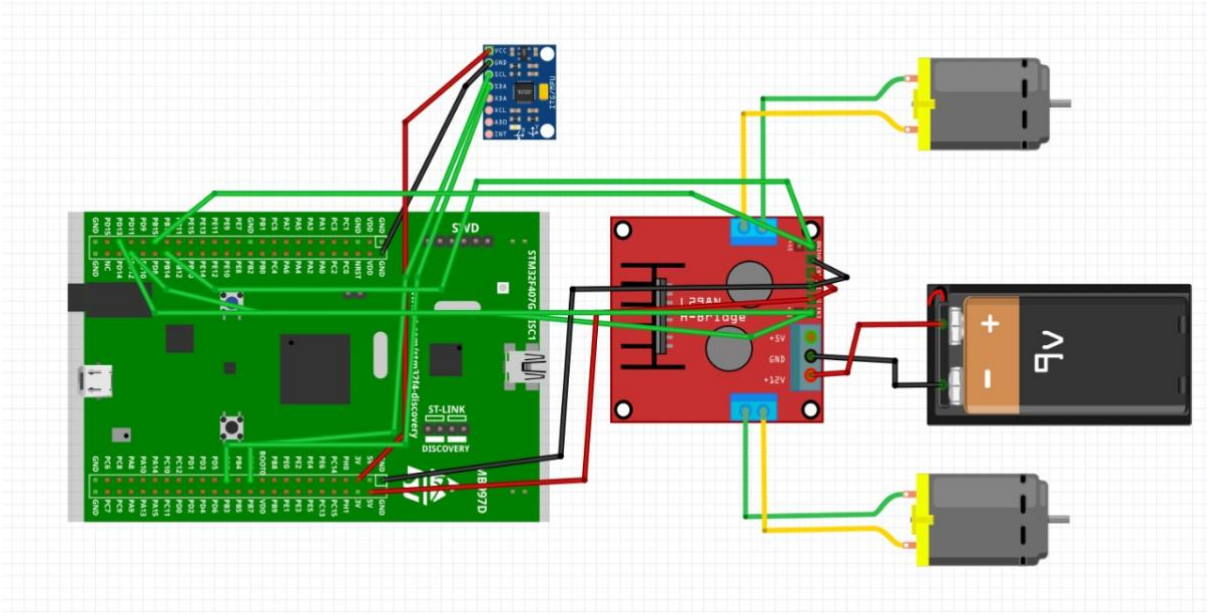
We have used 3 ports of the stm32

1)Port B for communicating with the MPU6050 sensor using I2C

2)Port A connected to L289N motor driver for controlling the direction of the motors

3)Port D connected to L289N motor driver for controlling the speed of the motor.

Circuit Connections:



PORT B : (for the I2C)

The pins PB6 and PB7 are used for the I2C communication between the stm32 and the MPU6050.

PB6 is connected to the SCL and PB7 to the SDA pin of the MPU6050.

Pin Configuration:

The pins PB6 and PB7 are set to

- Alternate function mode as they are used for I2C.
- Open drain and pull up mode
- High speed mode

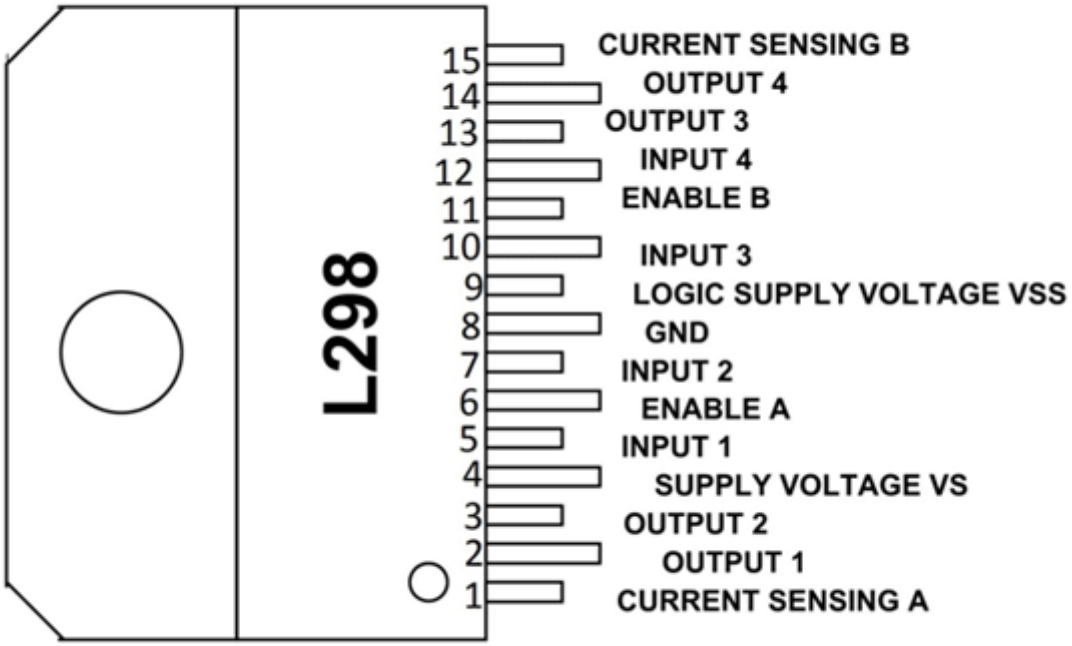
Port A :

The pins A1,A2,A3 and A4 are used

Connected to the L289N IC for controlling the direction of the motor.

PA1-> IN1 PA3->IN3

PA2->IN2 PA4->IN4



Configurations :

General purpose output mode

Pull down mode

High speed

Port D:

PD12,PD13,PD14 and PD15 pins are used as PWM output pins connected to the L289N enable pins

Configuration:

Alternate function mode for PWM

Pull up and High speed mode

PWM Configurations

Timer 4 is used for generating the PWMs

The system clock frequency of 84MHz is prescaled to drive the Timer 4 at 21MHz frequency

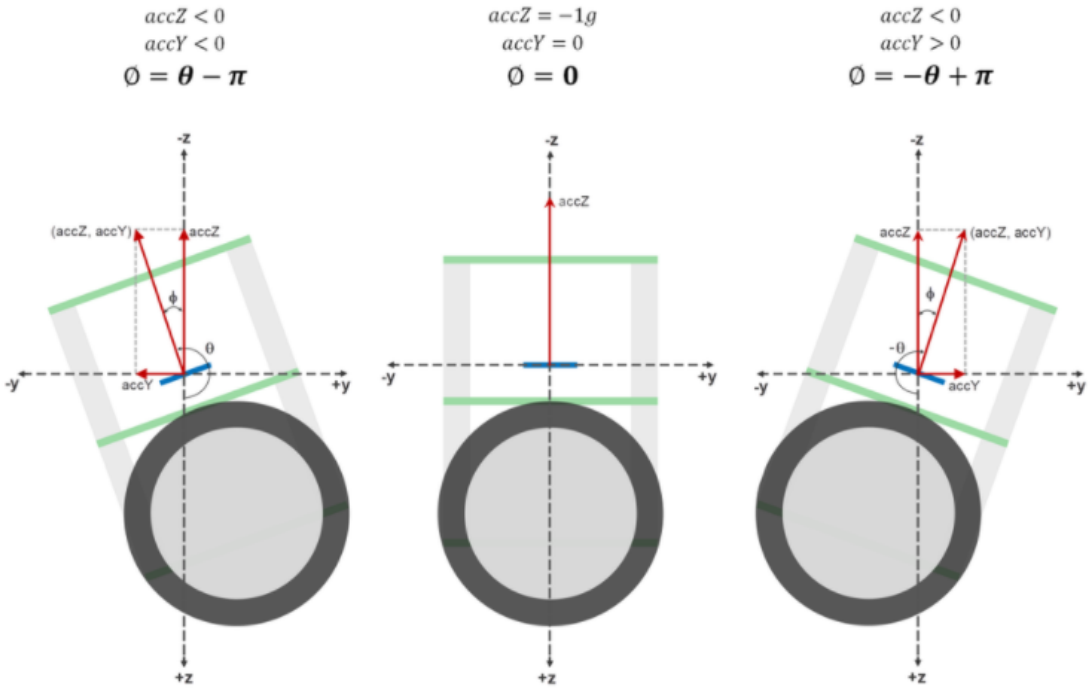
The frequency of the PWM is kept 20kHz by keeping the ARR value as 1050

Total 4 capture compare channels of the Timer 4 is used for generating PWMs with 4 different duty cycles.

MAIN AGORITHM:

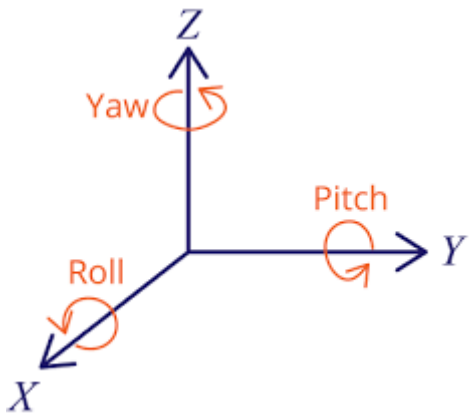
The accelerometer readings are used to calculate the angular deviation of the robot from the positive Z axis.

ANGLE CALCULATION USING ACCELERATION



Similarly the gyroscope readings are used to find the angular deviation of the robot from the positive Z axis.

ANGLE CALCULATION USING ANGULAR VELOCITY:



The gyroscope is read only for the Roll value that is the angular velocity about x- axis because that is sufficient for finding the angle the robot makes with the Z axis.

The angle is calculated as (Roll * time period) to give the angle made by the robot in the time 1 time period of the complimentary filter explained below.

Both these values are combined together with the help of a complimentary filter

$$\text{currentAngle} = \alpha.(\text{previousAngle} + \text{gyroAngle}) + (1-\alpha).(\text{accAngle})$$

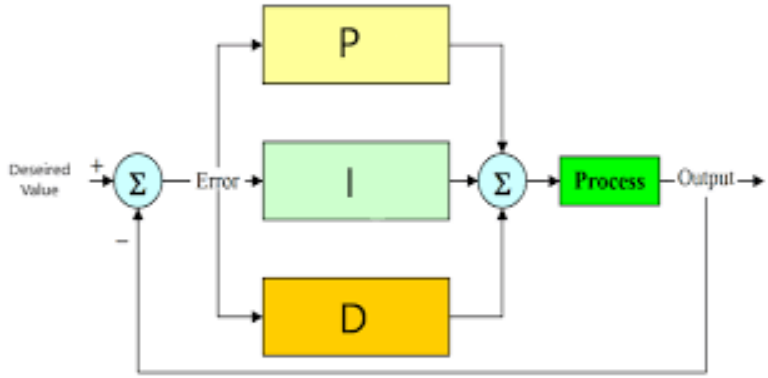
HPF LPF

Error Calculation

Once the angle of the robot from the Z axis is calculated we subtract it from the reference angle to find the error.

The error is then passed through the PID controller function.

PID CONTROLLER FUNCTION:



The output of PID controller is taken as the duty cycle that needs to be given to the PWM signals for controlling the speed of the motor such that the error reduces.

The direction of the motor to be driven is understood by the sign of the error calculated.

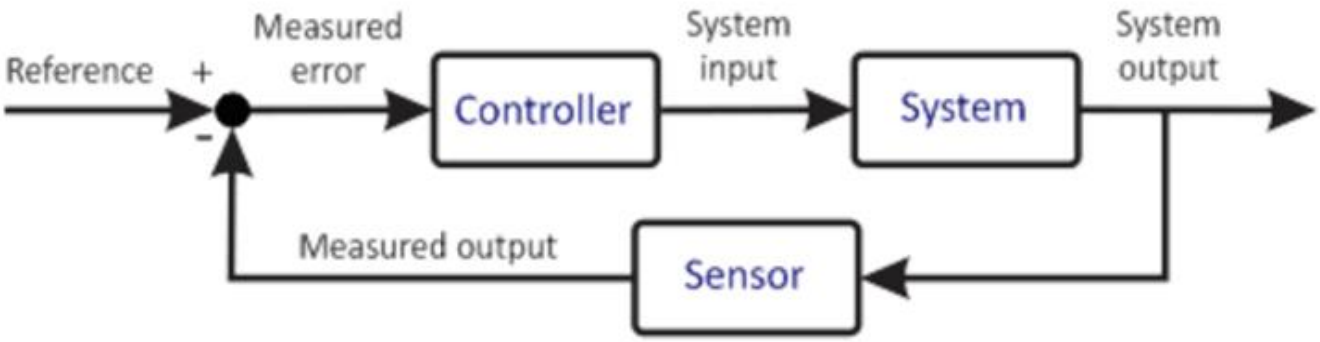
A negative error means the robot is falling in the backward direction.

A positive error means the robot is falling in the forward direction.

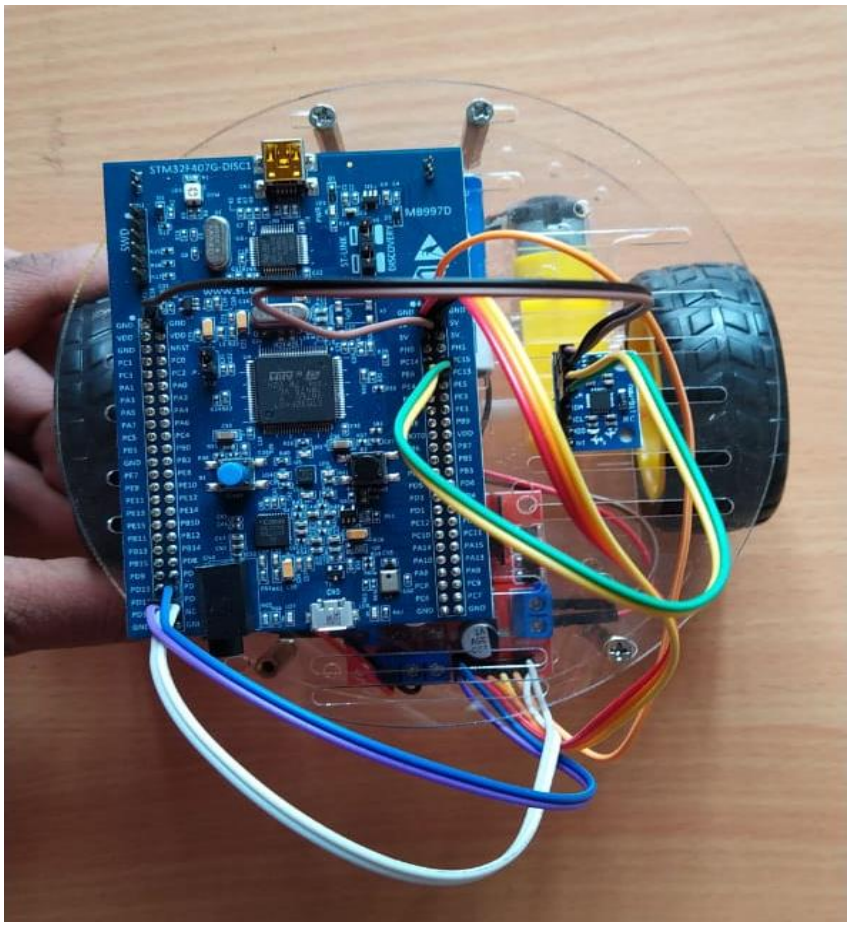
So for the robot falling forward the robot should be driven in the forward direction

If it is falling backwards the robot needs to be driven backwards. Thereby maintaining the centre of gravity above the pivot point so that the robot stays balanced and does not fall.

Design of Self balancing Robot system



Hardware Implementation :



SOFTWARE IMPLEMENTATION

REGISTER LEVEL PROGRAMING :

```
#include "stm32f4079xx.h"

#include <math.h>
#define MPU6050_ADDR 0xD0
#define SMPLRT_DIV_REG 0x19
#define ACCEL_CONFIG_REG 0x1C
#define ACCEL_XOUT_H_REG 0x3B
#define PWR_MGMT_1_REG 0x6B
#define WHO_AM_I_REG 0x75
#define GYRO_XOUT_H_REG 0x43
float ref  = 0;
float Kp=0.05,Kd=40,Ki=40;
float sum_of_errors=0;
float dt = 0;
float deg=57.3;
float Acc_angle=0;
float Gyro_angle=0;
float angle=0;
float error=0,duty=0;
#define INT_ENABLE_REG 0x38
int16_t x_read=0,y_read=0,z_read=0;
int16_t Gyro_X_RAW = 0;
int16_t Gyro_Y_RAW = 0;
int16_t Gyro_Z_RAW = 0;
int16_t Accel_X_RAW = 0;
int16_t Accel_Y_RAW = 0;
int16_t Accel_Z_RAW = 0;
float Ax, Ay, Az, Gx, Gy, Gz;
uint8_t check,temp;
int flag=0;
void EXT_Init(void)
{

//1. Enable TIM@ and GPIO clock
        RCC->APB1ENR |= (1<<0);  // enable PORTA clock
        RCC->APB2ENR |=(1<<14);
        RCC->APB2ENR |= (1<<0);
        SYSCFG->EXTICR[0] &= ~(0xf<<0);
        EXTI->RTSR |= (1<<0);  // Enable Rising Edge Trigger for PA0
        EXTI->FTSR &= ~(1<<0);  // Disable Falling Edge Trigger for PA0
        EXTI->IMR |=(1<<0);
        NVIC->ISER[0] |= 1<<6;
        NVIC_SetPriority (EXTI0_IRQn, 1);  // Set Priority
        NVIC_EnableIRQ (EXTI0_IRQn);  // Enable Interrupt
```

```
}
void TIM_Config()
{
	// Basic timer configuration
	RCC->APB1ENR |= (1UL << 2);    //Enable TIM4 clock
	TIM4->CR1 &= 0x1101;                    //Set the mode to Count up
	TIM4->CR1 |= 1UL; //Start the timer
}
void PWM_configuration()
{		TIM4->PSC = 1-1;          //PRESCALER = sys_clck/required_freq = 16MHz/16Mhz=1;
		TIM4->ARR = 1000;          // for a 16kHz PWM signal for no noise

		//Channel 1 config
		TIM4->CCMR1 |= (6UL << 4);    //Set OutputCompare mode to PWM
		TIM4->CCMR1 &= ~(3UL << 0);   //CC to output
		TIM4->CCER &= ~(1UL << 1);      //Output Compare polarity to active high
		TIM4->CCER |= (1UL << 0);        //Capture compare output enable

		//Channel 2 config
		TIM4->CCMR1 |= (6UL << (4+8));    //Set OutputCompare mode to PWM
		TIM4->CCMR1 &= ~(3UL << 8);   //CC to output
		TIM4->CCER &= ~(1UL << (1+4));           //Output Compare polarity to active high
		TIM4->CCER |= (1UL << 4);        //Capture compare output enable

		//CHANNEL 3

		TIM4->CCMR2 |= (6UL << (4));    //Set OutputCompare mode to PWM
		TIM4->CCMR2 &= ~(3UL << 0);   //CC to output
		TIM4->CCER &= ~(1UL << (1));    //Output Compare polarity to active high
		TIM4->CCER |= (1UL << 0);        //Capture compare output enable

				//CHANNEL 4

		TIM4->CCMR2 |= (6UL << (4+8));    //Set OutputCompare mode to PWM
		TIM4->CCMR2 &= ~(3UL << 8);   //CC to output
		TIM4->CCER &= ~(1UL << (1+4));           //Output Compare polarity to active high
		TIM4->CCER |= (1UL << 4);        //Capture compare output enable

}
void GPIO_configuration()
{	// GPIOB - for I2C connection  with MPU6050
	RCC->AHB1ENR |=(1UL<<1);//Enable clock for port B
	GPIOB->MODER |=(2UL<<12);//PB6 to alternate function pin
	GPIOB->MODER |=(2UL<<14);//PB7 to alternate function pin
	GPIOB->PUPDR |=(0x5UL<<12);//set PB6 and 7 as pull up pins
	GPIOB->OTYPER |=(0x3UL<<6);//Set PB6 and 7 as open drain
	GPIOB->OSPEEDR |=(0xAUL<<12);//Set PB6 and 7 as high speed
	GPIOB->AFR[0] |= (0x44<<24);//Set PB6 and 7 to alternate function 4
```

```
	// GPIOA - for direction control
	RCC->AHB1ENR|=RCC_AHB1ENR_GPIOAEN;  //ENABLED THE CLOCK FOR GPIOA
	GPIOA->MODER |=(0x55UL<<2); // GPIOA pins - 1,2,3,4 mode selected as general purpose
output
	GPIOA->PUPDR|=(0xAA<<2);// GPIOA pins-> 1,2,3,4 SET TO PULL down MODE
	GPIOA->OSPEEDR|=(0xAA<<2);//GPIOA pins 0,1,2,3 set to high speed

	// GPIOD - for PWM output
	RCC->AHB1ENR|=RCC_AHB1ENR_GPIODEN;
	GPIOD->MODER|=(0xAAUL<<24); //MODE OF PINS 12,13,14,15 of GPIOD SET TO AF
	GPIOD->PUPDR|=(0x55UL<<24);// GPIOD 12 and 13,14,15 set to pull up mode
	GPIOD->OSPEEDR|=(0xAAUL<<24); // speed of GPIOD 12 and 13,14,15 set to high speed

}
void I2C_Config(void)
{
	RCC->APB1ENR |=(1UL<<21);//Enable I2C clock
	I2C1->CR1 |= (1UL<<15);//Reset I2C
	I2C1->CR1 &= ~(1UL<<15);
	I2C1->CR2 |=(16UL<<0);//Set peripheral clock at 16MHz
	I2C1->OAR1 |=(1UL<<14);//Should be set high
	I2C1->CCR |=(0x50UL<<0);//Set SCL as 100KHz
	I2C1->TRISE |=(17UL<<0);//Configure maximum rise time
	I2C1->CR1 |= (1UL<<0);//Enable I2C
}
void I2C_Start (void)
{
	I2C1->CR1 |= (1<<10);//Enable the ACK Bit
	I2C1->CR1 |= (1<<8);//Send the start bit
	while (!(I2C1->SR1 & (1<<0)));//Wait for SB bit to set
}
void I2C_Write(uint8_t data)
{
	while (!(I2C1->SR1 & (1<<7)));//Wait till TX buffer is empty
	I2C1->DR = data;//Write data to I2C slave
	while (!(I2C1->SR1 & (1<<2)));//Wait till Byte transfer is completed
}
void I2C_Address (uint8_t Address)
{
	I2C1->DR = Address;  //  send the slave address
	while (!(I2C1->SR1 & (1<<1)));  // wait for ADDR bit to set
	temp = I2C1->SR1 | I2C1->SR2;  // read SR1 and SR2 to clear the ADDR bit
}
void I2C_Read (uint8_t Address, uint8_t *buffer, uint8_t size)
{
	int remaining = size;
	if (size == 1)
	{
```

```
		I2C1->DR = Address;  //  send the address
		while (!(I2C1->SR1 & (1<<1)));  // wait for ADDR bit to set
		I2C1->CR1 &= ~(1<<10);  // clear the ACK bit
		temp = I2C1->SR1 | I2C1->SR2;  // read SR1 and SR2 to clear the ADDR bit.... EV6
condition
		while (!(I2C1->SR1 & (1<<6)));  // wait for RxNE to set
		buffer[size-remaining] = I2C1->DR;  // Read the data from the DATA REGISTER
	}
	else
	{
		I2C1->DR = Address;  //  send the address
		while (!(I2C1->SR1 & (1<<1)));  // wait for ADDR bit to set
		temp = I2C1->SR1 | I2C1->SR2;  // read SR1 and SR2 to clear the ADDR bit
		while (remaining>2)
		{
			while (!(I2C1->SR1 & (1<<6)));  // wait for RxNE to set
			buffer[size-remaining] = I2C1->DR;  // copy the data into the buffer
			I2C1->CR1 |= 1<<10;  // Set the ACK bit to Acknowledge the data received
			remaining--;
		}
		while (!(I2C1->SR1 & (1<<6)));  // wait for RxNE to set
		buffer[size-remaining] = I2C1->DR;
		I2C1->CR1 &= ~(1<<10);  // clear the ACK bit
		I2C1->CR1 |= (1<<9);  // Stop I2C
		remaining--;
		while (!(I2C1->SR1 & (1<<6)));  // wait for RxNE to set
		buffer[size-remaining] = I2C1->DR;  // copy the data into the buffer
	}
}
void I2C_Stop (void)
{
	I2C1->CR1 |= (1<<9);  // Stop I2C
}
void MPU_Write (uint8_t Address, uint8_t Reg, uint8_t Data)
{
	I2C_Start ();
	I2C_Address (Address);
	I2C_Write (Reg);
	I2C_Write (Data);
	I2C_Stop ();
}
void MPU_Read (uint8_t Address, uint8_t Reg, uint8_t *buffer, uint8_t size)
{
	I2C_Start ();
	I2C_Address (Address);
	I2C_Write (Reg);
	I2C_Start ();
	I2C_Read (Address+0x01, buffer, size);//To read, set LSB to 1
```

```
	I2C_Stop ();
}
void MPU6050_Init (void)
{
	uint8_t check;
	uint8_t Data;
	MPU_Read (MPU6050_ADDR,WHO_AM_I_REG, &check, 1);//Check device ID

	if (check == 104)  // 0x68 will be returned by the sensor
	{
		Data = 0;
		MPU_Write (MPU6050_ADDR, PWR_MGMT_1_REG, Data);//Power up the sensor
		Data = 0x07;
		MPU_Write(MPU6050_ADDR, SMPLRT_DIV_REG, Data);//Sampling rate of 1KHz
		Data = 0x00;
		MPU_Write(MPU6050_ADDR, ACCEL_CONFIG_REG, Data);//accelerometer
range=+/- 2g
		Data=0x01;
		MPU_Write(MPU6050_ADDR, INT_ENABLE_REG, Data);//Enable interrupt when
data ready
	}
}
void read_acc(void)
{
	uint8_t Buf[6];
	// Read 6 BYTES of data starting from ACCEL_XOUT_H register
	MPU_Read (MPU6050_ADDR, ACCEL_XOUT_H_REG, Buf, 6);
	Accel_X_RAW= (int16_t)(Buf[0] << 8 | Buf[1]);
	Accel_Y_RAW = (int16_t)(Buf[2] << 8 | Buf[3]);
	Accel_Z_RAW = (int16_t)(Buf[4] << 8 | Buf[5]);
	Ax = Accel_X_RAW/16384.0;
	Ay = Accel_Y_RAW/16384.0;
	Az = Accel_Z_RAW/16384.0;
}
void read_gyro()
{
	uint8_t Rx_data[6];
	MPU_Read(MPU6050_ADDR, GYRO_XOUT_H_REG, Rx_data, 6);
	Gyro_X_RAW =(int16_t)(Rx_data[0]<<8 | Rx_data[1]);
	Gyro_Y_RAW =(int16_t)(Rx_data[2]<<8 | Rx_data[3]);
	Gyro_Z_RAW =(int16_t)(Rx_data[4]<<8 | Rx_data[5]);
	Gx = Gyro_X_RAW/131.0;  // FS_SEL = 0. So dividing by 131.0
	Gy = Gyro_Y_RAW/131.0;
	Gz=  Gyro_Z_RAW/131.0;
}

void EXTI0_IRQHandler( )
{
```

```
	EXTI->PR |= (1<<0);//Clear interrupt flag
	flag=1;
}
void Delay(uint32_t number)
{
	for(uint32_t i=0;i<number*500;i++)
	{
		__NOP();
	}
}
int main()
{
	TIM_Config();
	GPIO_configuration();
	PWM_configuration();
	I2C_Config ();
	MPU6050_Init ();
	EXT_Init();
	while (1)
	{
		if(flag==1){
			Delay(40);// require a delay of 40ms because time constant of filter is set to
40ms
			dt = 40; // time constant of complimentary filter= 40ms
			read_acc();
			Acc_angle = atan2(Ay,Az)*deg;
			read_gyro();
			Gyro_angle=Gz*dt*deg ;
			angle=0.99*(Gyro_angle)+0.01*(Acc_angle);
			error=ref-angle;
			sum_of_errors+=error*dt;
			duty=(Kp*error + (Kd*1000*(error)/dt)+Ki*sum_of_errors/1000)/100000;
			if(duty<0)
			duty=-duty;
			if(error>0)// robot falling forward
			{
				// move robot forward

				GPIOA->BSRR |=(0x1<<18)|(0x1<<20); // clr pins A2 and A4 i.e IN1
and IN3 of L289n
					GPIOA->BSRR|=((1<<1)|(1<<3)); // set pins A1 and A3 i.e IN1
					//and IN3 of L289N // IN1=1 and IN2=0

				// IN3=1 and IN4=0

				TIM4->CCR2=0;	//13....away from arrow.Motor 2
				TIM4->CCR4=0;	//15....along arrow....Motor1
				TIM4->CCR3=duty;	//14...along arrow...Motor2
```

```
                        TIM4->CCR1=duty+250;      //12...away from arrow.Motor 1

                }
                else if(error<0) // robot falling backwards
                {
                        // move robot backwards
                        GPIOA->BSRR |=(0x1<<17)|(0x1<<19); // clr pins A1 and A3 i.e IN1
and IN3 of L289n
                        GPIOA->BSRR |=(0x1<<2)|(0x1<<4); // set pins A2 and pins A4 i.2
IN2 and IN4 of L289n

                // IN1=0 and IN2=1

                // IN3=0 and IN4=1

                        TIM4->CCR2=duty;      //13....away from arrow.Motor 2
                        TIM4->CCR4=duty+250;  //15....along arrow....Motor1
                        TIM4->CCR3=0;    //14...along arrow...Motor2
                        TIM4->CCR1=0;      //12...away from arrow.Motor 1

                }
                else if(error==0){ // robot in mean position
                        // stop the motor
                        TIM4->CCR1=0;
                        TIM4->CCR2=0;
                        TIM4->CCR3=0;
                        TIM4->CCR4=0;
                        GPIOA->BSRR=0;
                        GPIOA->BSRR=0;
                }

                        flag=0;
        }
        //Delay(1000);

    }
}
```

Result Analysis:

The results were satisfying as we were able to get the readings from the MPU6050 which means the I2C communication was established between the microcontroller and the MPU06050.

Then the processing of these values through code also gave us the desired values which we were expecting. Hence the code written was perfect.

The initial testing of the project gave us a lot of errors in terms of the robot not being able to balance as we would want. The various parameters were then adjusted for making the output function ito give a desired behaviour.

Conclusion:

Working on a project with STM32F407 requires lot of hard work. The vast amount of registers that need to be configured before you can work on the main code can be very intimidating but then once you get a flow of the registers of STM32 the configuration becomes an easy step and an enjoyable one too.

Working on a project like this gave us lot of hands on experience and debugging skills. The completion of the project gave us immense satisfaction and motivation to work on more such projects in future.

Acknowledgement:

We would like to thank or Professor Selvakumar K who guided us in this project and pushed us to do a project like this,

Reference:

[1] Wu Junfeng , Zhang Wanying, Research on Control Method of Two-wheeled Self-balancing Robot
https://sci-hub.ee/10.1109/ICICTA.2011.132

[2] Ji-Hyun Park, Baek-Kyu Cho Development of a self-balancing robot with a control moment gyroscope
https://journals.sagepub.com/doi/full/10.1177/1729881418770865

[3] HELLMAN, HANNA SUNNERMAN, HENRIK, Design and control based on the concept of an inverted pendulum
http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A916184&dswid=-140

[4] midhun_s, Arduino Self-Balancing Robot

https://www.instructables.com/Arduino-Self-Balancing-Robot-1/
[5]MPU6050 REFERCNE MANUAL AND DATASHEET
https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf
[6]REFERENCE MANUAL OF L289N
https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf

[7]DATASHEET OF STM32F07VG
https://datasheetspdf.com/datasheet/STM32F407VG.html