SUMMER RESEARCH INTERNSHIP

IN

EMBEDDED SYSTEMS

SELF BALANCING BOT

Submitted by

ATHARVA MHETAR	16EC123
SAPTARSHI MONDAL	16EC124
RAKSHANA GUNASEKARAN	17EC118
JATIN J M	16EC219

Under the Guidance of

Prof. HARESH DAGALE

Principal Research Scientist, IISc, Bangalore

DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING
INDIAN INSTITUTE OF SCIENCE,
BANGALORE
INDIA

JULY 2019

CONTENTS

INTRODUCTION

This project aims at building a two wheel self-balancing robot based on the concept of inverted pendulum and the application of a PID control system. The following section contains the origin and background of this system and furthermore the scope of the project. The project was carried out as a part of the summer internship program at DESE, IISc Bangalore.

BACKGROUND

Self-balancing bots are robotic devices that are capable of maintaining an upright position on two wheels. This is achieved by collecting data from the accelerometer and gyroscope and using control algorithms to balance the bot. This project is inspired by the Segway, which makes use of movements by the body for steering purposes where the centre of mass is located above the pivot.

The bot was implemented using the Tiva™ C series TM4C123GH6PM microcontroller from Texas Instruments. Offset calibration was done by trial and error methods by serially displaying the values from sensors through UART. PID tuning was performed through Ziegler-Nichols method. Moreover, I2C communication was used to receive values from MPU6050 sensor and PWM signals were generated accordingly to control the motors.

SCOPE

This project includes the mechanical design overview and the system requirements for building a self-balancing bot. The Tiva™ C series TM4C123GH6PM microcontroller was programmed in Code Composer Studio (IDE from TI) using embedded C. The current functionality of the bot is to balance on its own. By adding more functionalities the bot can be made to move in different directions. The concepts of UART, I2C communication, PID control, PWM and Complementary filters were applied in the implementation.

THEORY

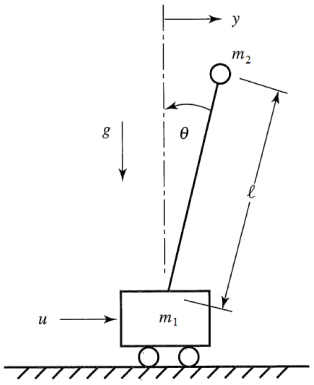
OVERVIEW

In the recent years, Segways have emerged into the market. These are basically two wheeled self-balancing bots that make use of bodily movements for steering. In these applications the centre of mass of the system is located above their pivot points, thus requiring active control for balance. The following project thus focuses on working on a similar idea of making a self-balancing bot using the concepts of control theory.

Segway personal transporter

THEORY OF INVERTED PENDULUM

The inverted pendulum is a classic problem in dynamics and control theory and is used as a benchmark for testing control strategies. It can be thought of as a pendulum that has its centre of mass above its pivot point. It is unstable and without any additional help will fall over. However, it can be suspended stably in this inverted position by using a control system to monitor the angle of the pole and move the pivot point horizontally back under the centre of mass when it starts to fall over, keeping it balanced. For convenience the pendulum's degrees of freedom are limited to one direction, the angle Θ moving in the xy-plane.



Inverted pendulum diagram

Refer this link for more information: https://en.wikipedia.org/wiki/Inverted_pendulum

FILTERS

The filters play an important role in estimation of the angle to which the bot has titled. Now these readings can be obtained directly from the sensor mounted on the bot. However they are not very accurate to make the tilt angle estimation. This is because of the presence of noise and gyro drift. The MPU6050 which is a 6 axis sensor gives the accelerometer and gyroscope readings in the x, y and z directions. The accelerometer measurements are noisy in general, whereas the gyroscope drifts over time and the readings from it do not prove useful. A solution to this is fusing both the accelerometer and gyroscope values and this can be done with the use of filters. In control theory, complementary filters are widely used for this purpose as its implementation is simple. Kalman filter involves high computations, but gives more accurate results. However implementing a Kalman filter is quite challenging and difficult. In this project we implemented both filters and found that Kalman proved computationally heavy. As a result the final implementation contains the use of complementary filter.

COMPLEMENTARY FILTER

The idea behind complementary filter is to take less fluctuating (slow moving) values from accelerometer and tilt sensitive (fast moving) values from a gyroscope and combine them. Accelerometer gives a good indicator of orientation in static conditions. Gyroscope gives a good indication of tilt in dynamic conditions. So the

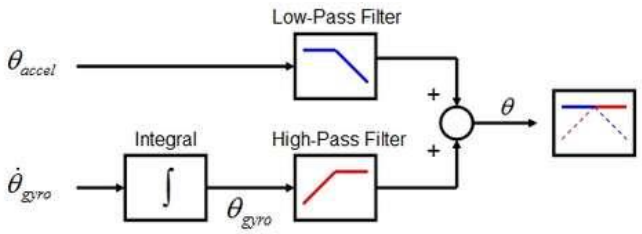
idea involves, passing the accelerometer signals through a Low pass filter and the gyroscope signals through a High pass filter and combine them to give the final tilt. The equation of complementary filter is given by:

Angle = (1-α)(angle + gyro*dt) + (α)*(acc)*

First reading is the angle obtained from gyroscope integration and the second is the one from accelerometer. On careful observation it can be noticed that the filter is nothing but a weighted average of the values from accelerometer and the gyroscope with the weights α and 1- α.

How to choose α?

α = (τ / τ + dt) where τ is the desired time constant i.e. how fast we wish the readings to respond and dt = 1/fs where fs is the sampling frequency. We have used the value of α = 0.02 in the project.



Complementary filter block diagram

For detailed information please refer:
https://sites.google.com/site/myimuestimationexperience/filters/complementary-filter

KALMAN FILTER

Kalman filtering is also known as linear quadratic estimation (LQE). It uses a system's dynamic model, known control inputs to that system, and multiple sequential measurements (from sensors) to form an estimate of the state of the system. It is a common sensor fusion and data fusion algorithm. It makes use of information of previous state of the system and present input measurements to predict the next state of the system. In simple terms it involves statistical measurements of the position and velocity of the system in our case to estimate the next state of the system. The position and velocity measurements of the system being probable, include noise which has a Gaussian characteristic.

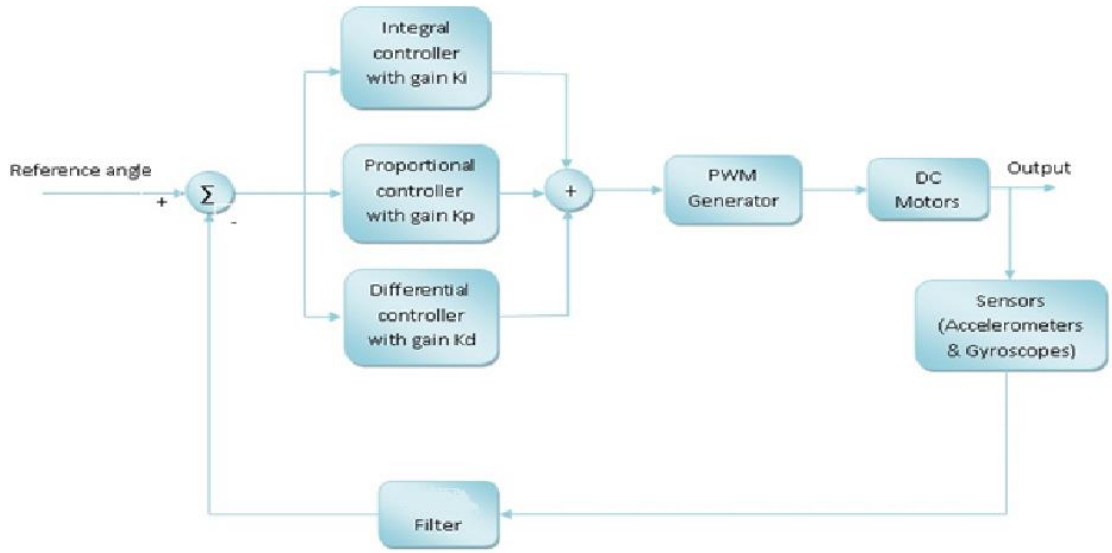
Refer the following links for mathematical derivations to understand the Kalman filter in detail:

http://web.mit.edu/kirtley/kirtley/binlustuff/literature/control/Kalman%20filter.pdf

https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/

http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/

PID CONTROL SYSTEM



The above figure shows the block diagram for the overall system along with the PID control block. As we wish for the bot to stay in an upright position, the reference angle i.e. the set-point is angle zero. The error is evaluated as the difference of present angle and previous angle and this error acts as an input to the PID control block. This is an example of closed loop control system also referred to as the negative feedback system. The PID control then outputs a correction factor as

Correction = Kp * present error + Ki * total error + Kd * (present error – previous error)

The error is used on the PID controller to execute the proportional term, integral term is used for reduction of steady state errors, and the derivative term is used to handle overshoots. In simple words, the first term is used to calculate scaled correction, the second term forces the bot to move towards the mean position faster whereas the third term resists sudden change in deviation.

Note that, Kp, Ki and Kd are constants which are set experimentally.

Tips for faster PID tuning

1. Set the Ki and Kd to zero, then adjust Kp so that the bot starts to oscillate about the balance position. Kp should be large enough for the bot to move but not too large else the movement might not be smooth.
2. Once Kp is set, increase Kd such that the bot is able to recover quickly to the balance point after giving a slight push. Although, Kd should not be too high otherwise it leads to jitters in the movement of the bot.
3. Finally when both Kp and Kd is set, increase Ki such that the bot accelerates faster when off balance.

Refer this link for more information: https://www.youtube.com/watch?v=uyHdyF0_BFo

DESIGN SPECIFICATIONS

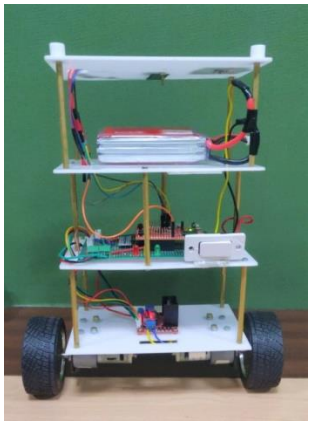
HARDWARE

MECHANICAL DESIGN

The main body of the bot comprises of four 2mm thick acrylic sheet levels supported by brass hexagonal spacers. We used acrylic sheets as they are light, insulating and do not interfere with the circuitry mounted on them. The structure was constructed by drilling holes on the platform and were tightened using screws. The first layer consists of the motor driver followed by the second layer containing the circuit board and the power switch. Subsequently, the battery is attached to the third layer and finally the IMU sensor is mounted on the topmost platform. 12v DC motors were

initially used in this model which were upgraded in the second model. At last the bot was covered using an acrylic case for clean presentation purposes. The following table enlists all the mechanical components that were required.

1	MPU6050 sensor (3 axis accelerometer, 3 axis gyroscope)
1	TIVA^{TM} C series TM4C123GH6PM MCU from Texas Instruments
1	L298N motor driver
2	12v DC motors (6mm shaft diameter)
2	L Clamps for motors
1	11.1 V (3 cell) LiPo battery, charger and switch
1	7805 voltage regulator
2	Level Shifters for 3.3V to 5V logic shift
3	2 pin terminal connectors
1	5 pin and 6 pin RMC connectors
2	80 mm diameter wheels
24	5cm long; 3mm screw size Hexagonal brass (male and female) spacers
2	12mm Hex Wheel Adapter for 6mm Shaft
4	18cm X 11cm Acrylic sheets for chassis of the bot
1	General purpose board
50	3mm bolts and nuts
5	female to female jumper wires
1	rolls of single strand connection wires



The second version of the bot was made using the same circuit connections but with alterations in the mechanical design. We used 2 layers of 5mm thick arcylic sheet supported by hexagonal brass spacers. The first layer comprised of the circuit board, motor driver and the switch. The L clamps were attached to the under side of the chassis and the planetary motors of 8mm shaft were clamped on with 105mm

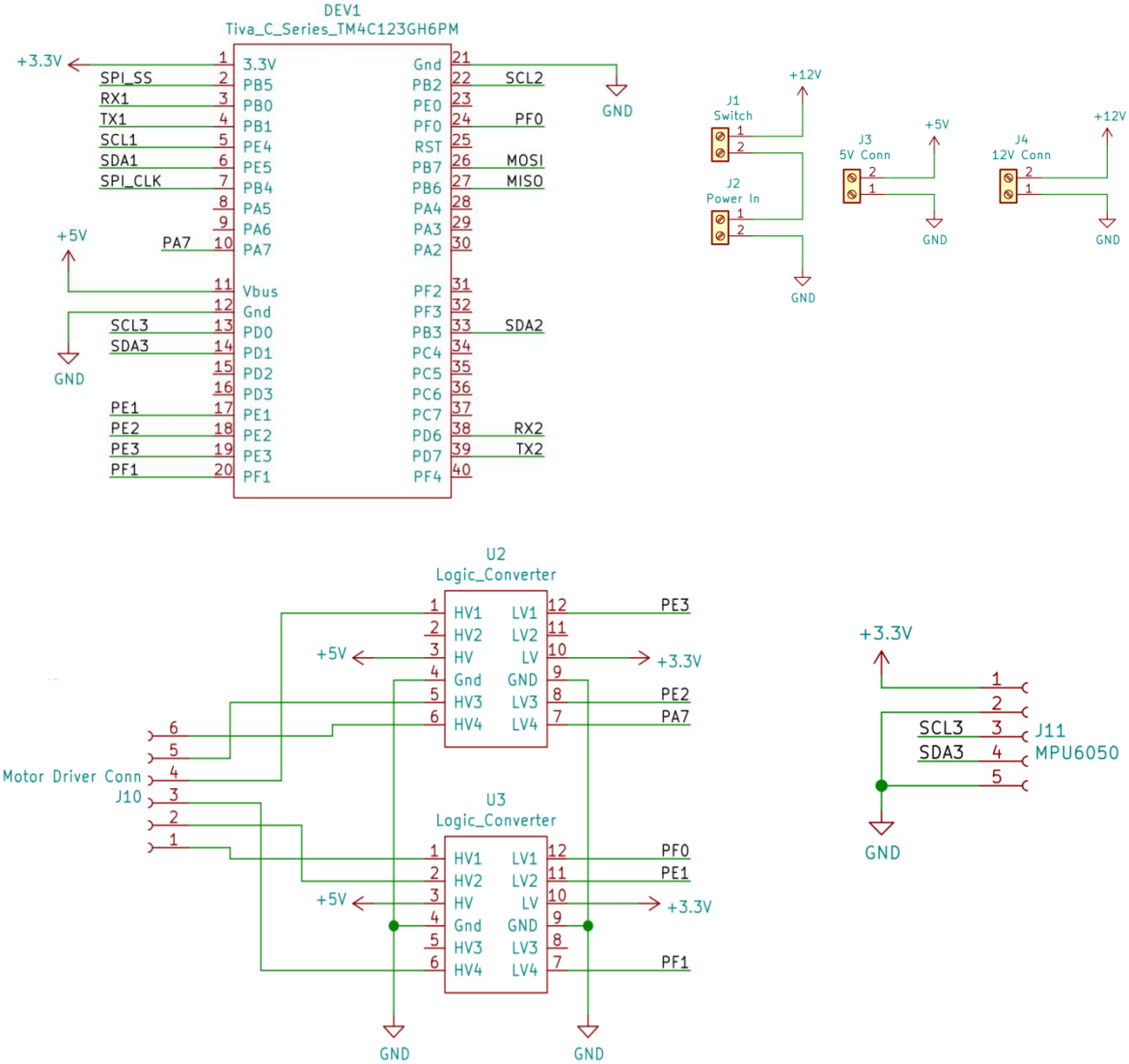
wheels. The second layer had the imu sensor mounted on along with an aluminium shaft of 40cm height to achieve a Segway like design.

Altered components

2	Orange Planetary Gear DC Motors 12V 262RPM 4.52Kg-CM-PG36M555-19.2K
2	L Clamps for planetary motors
2	105 mm diameter wheels
1	aluminium shaft
2	5mm thickness laser cut acrylic chassis

ELECTRICAL DESIGN

The following schematic was drawn using Saturn PCB Toolkit and it represents how the peripherals are connected to the microcontroller. The motor driver is connected to port E and port F and the mpu6050 uses the I2C 3 module. The input power is derived through a 2 pin switch and passed to an L7805 voltage regulator which gives an output voltage of 5V. Subsequently, the 5v and 12v supply are tapped out to the connectors. The connections are made according to the figure for the MPU6050 and the GPIO pins of 3.3v level are fed to a 3.3v to 5v logic converter. Finally, this output is given to the motor driver that controls the motors.

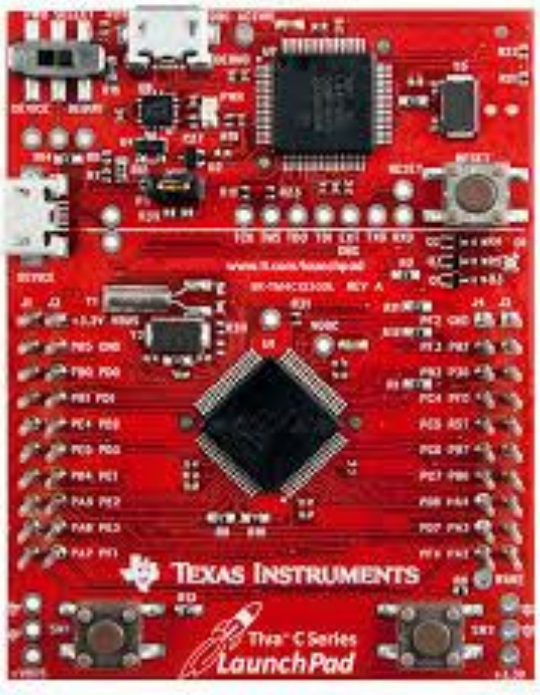


Components used:

Tiva™ TM4C123GH6PM MCU

Manufactured by Texas Instruments, this board is a high performance 32 bit, ARM® Cortex®-M4F-based microcontroller. It is targeted for industrial applications like remote monitoring, network appliances, factory automation, HVAC and building control, gaming equipment, motion control, transportation, and security.

Its features include an 80MHz processor core with system timer (SysTick), integrated nested vectored interrupt controller (NVIC), wake-up interrupt controller (WIC) with clock gating and memory protection unit (MPU).
Moreover, it has Advanced serial integration featuring eight UARTs with IrDA, 9-bit, and ISO 7816 support along with advanced motion control featuring eight pulse width modulation (PWM) generator blocks, each with one 16-bit counter, two PWM comparators, a PWM signal generator, a dead-band generator, and an interrupt/ADC-trigger selector.
Also provides analog support featuring two 12-bit analog-to-digital converters (ADC) with 12 analog input channels and a sample rate of one million samples/second, two analog comparators; 16 digital comparators, and an on-chip voltage regulator.



Tiva™ TM4C123GH6PM board

For more information please refer to:
http://www.ti.com/lit/ds/symlink/tm4c123gh6pm.pdf

MPU6050

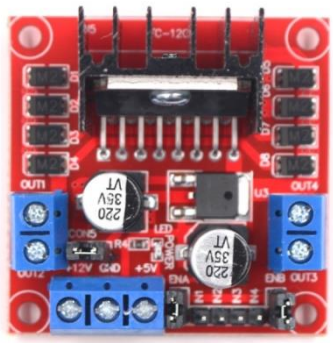
One of the most significant components in building a self-balancing bot is an accelerometer and gyroscope module. We have used this high precision, 3 axis gyroscope and 3 axis accelerometer to measure the angle of deviation. This devices are small in size and very cheap to buy. But, they share their own disadvantages. Gyroscopes are good for short term and quick movements but tend to drift over time as the error accumulates. Whereas accelerometer is good at sensing slower and more prolonged movements. Hence these two sensors need to be fused to read out accurate measurements using filtrations.
This modules uses I2C to communicate with the microcontroller and needs 3.3 v to input voltage. The measurement ranges for the accelerometer are +/-2g, +/-4g, +/-8g , +/-16g and for the gyroscope are +-250/500/1000/2000 dps.

MPU6050

For more information please refer to:
https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf

L298N Motor driver

The L298N driver is a high voltage, high current dual full bridge driver designed to accept the standard TTL logic levels and can drive inductive loads such as dc motors, stepper motors, relays and solenoids. Since the microcontroller operates at a low voltage and current whereas the motors requirements are higher, the driver acts as a current amplifier. The maximum supply voltage is 46v and maximum output dc current is 4a. It has practically all the features that are required in a good motor driver, including thermal-shutdown, as in it will slow down and stop if overloaded.

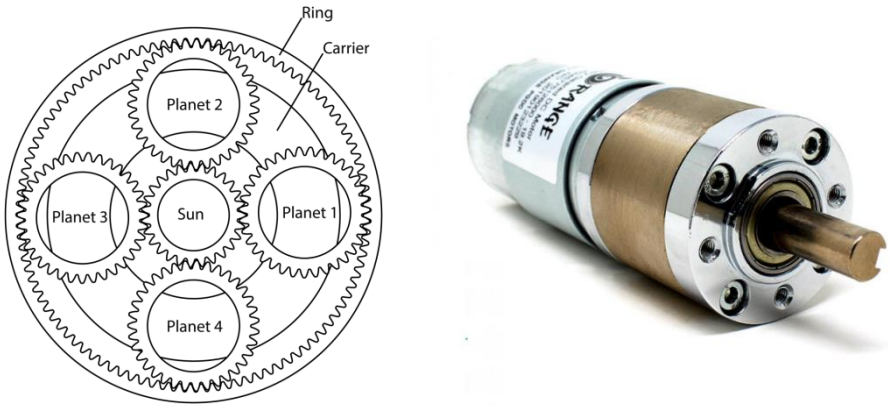


L298N motor driver

For more information please refer to :
https://www.mouser.in/datasheet/2/389/l298-954744.pdf

Orange Planetary Gear DC Motor

Planetary motors are used in applications which require high precision and reliability. The gear train consists of the sun gear, the planet carrier and the crown or ring. The sun gear or the central gear rotates of=n the central axis and is larger in size. The planet carrier holds up to 3 gears of the same size which mesh with the sun gear. Crown is an outer ring that meshes with the planets and contains the whole epicyclical train.



Planetary motor gear train

Due to its higher speed, radial and axial loads offer robustness that minimizes the misalignment of the gear. Moreover, the noise level are reduced since there is more surface of contact. With more teeth in contact the mechanism can transmit and withstand more torque uniformly. We have used Orange Planetary Gear DC Motor that requires 12v dc supply and has a reduction ratio of 19.2:1 producing an rpm of 262 rpm with 4.52kgcm torque.

SOFTWARE

Note: Kindly go through the following section parallel to datasheet of TM4C123GH6PM and MPU6050, as well as the source code for easy understanding.

Code Composer Studio (CCS) is an IDE from Texas Instruments that was used for programming the TIVA™ C series TM4C123GH6PM MCU used in this project. The following section focuses on certain parts of the code and how the datasheet of the MCU was used to program the interfacing of sensors, implementing the PID control system and generation of PWM to achieve the goal of the project

I2C (Inter-Integrated Circuit) Communication:

Key Points:

1. There can be multiple masters and slaves connected together while only a master and a slave can communicate at once over a channel.

2. Used to integrate devices of different speeds.

3. Each slave has a unique slave address which is used by master to communicate the data to a particular slave.

4. The data transfer from master to slave is serial and it is split into 8-bit packets.

5. It uses just 2 wires SCL (Serial Clock) and SDA (Serial Data).

6. Communication is initiated by master device which generates the start condition followed by slave address. There is a bit specifically assigned to mention the task being performed i.e. read (1) or write (0) by the master along with sending the slave address. Once all the bytes are read or written the master generates a stop condition indicating the end of communication.

Initialization of I2C

We have used I2C Module 3 which is a Port D peripheral (**PD0 -> SCL and PD1 -> SDA**). Here PD1 is Open Drain configured which means that it has no pull up register attached to it. Hence it is important to be explicitly declared in the code.

1. Enable clock to I2C and GPIO registers.
2. Since we are using the GPIO pins D0 and D1 from port D, for them to be used for I2C communication, the AFSEL (alternate function select) register needs to be enabled

3. Further, the PCTL (port control register) needs to be assigned a bit field encoding value associated with the I2C peripherals for GPIO Do and D1.

```
void I2C3_init(void)
{
    SYSCTL_RCGCI2C_R |= 0x08; /* enable clock to I2C3 */
    SYSCTL_RCGCGPIO_R |= 0x08; /* enable clock to GPIOD */

    /* PORTD 1 (SDA), 0 (SCL) for I2C3 */

    GPIO_PORTD_AFSEL_R |= 0x03; /* PORTD 1, 0 for I2C3  */ //D0=scl D1=sda
    GPIO_PORTD_PCTL_R &= ~0x000000FF; /* PORTD 1, 0 for I2C3 */
    GPIO_PORTD_PCTL_R |= 0x00000033;
    GPIO_PORTD_DEN_R |= 0x03; /* PORTD 1, 0 as digital pins */
    GPIO_PORTD_ODR_R |= 0x02; /* PORTD 1 as open drain */

    I2C3_MCR_R = 0x10; /* master mode */
    I2C3_MTPR_R = 7; /* 100 kHz @ 16 MHz */
}
```

I2C functionalities

MPU6050:

1.It has 6 degrees of freedom (3 axis gyroscope and 3 axis accelerometer) along with a temperature sensor.

2. Power Supply - 3 to 5 V

3. Communication - I2C Protocol

4. It has a built in 16bit ADC.

5. Configurable I2C address (0x68 or 0x69 decided by setting or resetting ADo pin of MPU).

Initialization of MPU6050

In here we are setting limits to 16bit ADC of the sensor as +/- 250dps and +/- 2g so as to get more precise values and being sure that values from sensor to our project will be within the limits. Look into the datasheet of MPU6050 to understand the below configuration.

I2C just has a wire using which we can configure the slave device. It is done by first sending the slave address which has to be configured followed by address of the register of slave we want to configure followed by the values to be written or read from the register.

```
void MPU6050_init(void)  // Why writing those values MDR_R ?  ### Link: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000
{
    I2C3_byteWrite(SLAVE_ADDR, 0x6B, 0x01); // clock 8 mhz pll with x axis gyro reference ???????????????
    delayMs(100);
    I2C3_byteWrite(SLAVE_ADDR, 0x68, 0x06); // signal path reset (resets the sensor path of accelerometer and gyroscope)
    delayMs(100);
    I2C3_byteWrite(SLAVE_ADDR, 0x6A, 0x00); // i2c_if_dis = 0  // resetting sensor registers of acc and gyro
    delayMs(100);
    I2C3_byteWrite(SLAVE_ADDR, 0x1A, 0x00); //fsync and dlpf disabled
    delayMs(100);
    I2C3_byteWrite(SLAVE_ADDR, 0x19, 0x07); //sample rate set to 1 khz   [(gyro_output_freq i.e.8khz)/(1 + sampler_div i.e. 7)]
    delayMs(100);
    I2C3_byteWrite(SLAVE_ADDR, 0x1B, 0x00); // +/- 250dps gyrometer  configuration
    delayMs(100);
    I2C3_byteWrite(SLAVE_ADDR, 0x1C, 0x00); // +/- 2g  accelerometer configuration
    delayMs(100);
}
```

UART (Universal Asynchronous Receiver/Transmitter)

Key Points:

1. Parallel data is converted to serial data at TX and vice versa at Rx.

2. No clock signal is required instead start, stop and parity bits are used to mark the beginning of transmission, end of data and the data format.

3. Rx sends and TX reads the bits at the rate of 115200 bits per second (bps).

4. Single master and single salve communication.

5. MSB bits are transmitted first.

Initialization of UART

In here we have used UART Module 0 which is a Port A peripheral. UART is being used as a debugging tool. Please refer to the source code for better understanding.

```
void init_uart(void)
{   //use masking for all the initializations if doesn't work

    SYSCTL_RCGC2_R |= 0x01;  //Clock and Enable Port A wrt functionalities
    SYSCTL_RCGCUART_R |= 0x01;   //Enable and provide Clock to UART0
    SYSCTL_RCGCGPIO_R |= 0x01;   //Clock to PORTA
    UART0_CTL_R |= 0x00; //Disable UART for setting purpose
    UART0_IBRD_R |= 0x08; //For 115200bps 8 = int [ system_clk i.e. 16MHz / (16 * baud_rate)]
    UART0_FBRD_R |= 0x2C; //For 115200bps 44 = int [(fractional part * 64) + 0.5]
    UART0_LCRH_R = 0x60; //for 8-bit data size, no FIFO, 1 stop bit, no interrupt, no parity
    UART0_CTL_R = 0x301; //for en, rxe, txe
    GPIO_PORTA_DEN_R |= 0x03; //PA0,PA1 digital IO
    GPIO_PORTA_AFSEL_R |= 0x03; //PA0,PA1 alternate functions rxe, txe
    GPIO_PORTA_PCTL_R |= 0x11; //PA0 and PA1 pins for UART function
    GPIO_PORTA_AMSEL_R |= 0x00; //To disable ADC (disabling analog functionality)

}
```

UART functionalities

UART values to be converted into strings so as to be printed on the screen. Reverse function is used as we have extracted the MSB bit and stored in LSB position. This is done as we are unaware of the size of the number present before decimal.

Interrupt and PID algorithm

Interrupt

An interrupt is a signal to the processor emitted by the hardware or software indicating that an event needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts execution of an Interrupt Service Routine (ISR) or Interrupt Handler which is a program written to guide the controller about what has to be done when an interrupt occurs.

In here, as there can be only one master and a slave at a time and as we have 2 slaves (Motor Driver and MPU6050) we need to regularly switch between the two. So here we have used interrupt to stop reading values from MPU6050 at every 20ms and then send the PWM to the motor driver.

```
void timer0A_init(void)
{
    SYSCTL_RCGCTIMER_R |= 1;
    TIMER0_CTL_R = 0;  // DISABLE TIMER
    TIMER0_CFG_R = 0x04;  //16 BIT MODE
    TIMER0_TAMR_R = 0x02;  // DOWN COUNT AND PERIODIC
    TIMER0_TAILR_R = 20000 - 1;  //LOAD VALUE
    TIMER0_TAPR_R = 16 - 1;  //PRESCALER VALUE
    TIMER0_ICR_R = 0x1;  // CLEARING TIMEOUT INTERRUPTS
    TIMER0_IMR_R = 0X00000001;  //TIMEOUT ENABLE
    NVIC_PRI4_R = (NVIC_PRI4_R & 0x8FFFFFFF) | 0X30000000;
    NVIC_EN0_R = 0x00080000;
    TIMER0_CTL_R |= 0x01; //ENABLE TIMER AND START COUNTING
    EnableInterrupts();
}
```

PID algorithm

It forms the link between estimation of the angle and controlling the motors i.e. generating the required PWM to balance the bot.

Duty = Kp * Present Error + Ki * Integral Error + Kd * Differential Error;

Here Kp, Kd and Ki are constants found by Trial and Error.

PID controller has 3 parts or components:

1. Present Error: It indicates the deviation of the bot from its stable position (upright position i.e. Set Point = angle zero) given by Set Point – Complementary Filter Angle.

2. Integral Error: It accounts for the increase in error generated if the bot isn't able to recover to stable position. The error gets added up indicating the need to increase PWM given by Total Error + Present Error.

3. Differential Error: It indicates the amount of angle which couldn't be recovered post last iteration given by

{Previous Complementary Filter Angle – Set Point} - {Complementary Filter Angle – Set Point}

```
void timer0_handler(void)
{
    int Kp = 2500, Ki = 5000, Kd = 30;        // 45,0.01,0.1 // ki = 500,1000,5000 // kd =30
    double present_error,dt = 0.02;   //dt=20msec which timer overflow interrupt
    TIMER0_ICR_R = 0x1;  //clear timeout

    yz = (double) sqrt(ayf * ayf + azf * azf);
    accel_angle = 0.8* accel_angle + 0.2 * atan2((double) (axf), yz) * 180 / 3.141592 ;

    prev_angle = angle_est;
    angle_est = ((0.99) * (angle_est - (double) (gyf * dt)) + (0.01 * accel_angle));   //Complementary filter


    Deviation = (prev_angle - set_point) - (angle_est - set_point);
    present_error = set_point - angle_est;

    //PID algorithm
    correction = Kp * (present_error) + Ki * Total_error + Kd * Deviation/dt;   //Deviation*50 = Deviation/dt
    Total_error = Total_error + present_error*dt;

}

void PID_Controller(void)
{

     if(angle_est>0){
               if(correction < 0){
               duty = correction;
               set_dutycycle1((-1)* (int)(duty));
               backward();
             }

     }

     else{
            if(correction > 0){
               duty =  correction;
               set_dutycycle1((int)duty);
               forward();
            }

         }
}
```

PWM

The idea of PWM involves modulating/varying only the width of the pulse, not its frequency. In the digital control system, PWM is used to control the amount of power sent to the device.

The speed of the motor depends on load, voltage and current. In our case since we have a fixed load we can maintain a steady speed of the motor using PWM. The amount of power supplied to the motor thus increases/decreases its speed. So wider the pulse, higher the speed.

PWM is thus a method to produce variable voltage using digital means.

Principle

Duty cycle and frequency are the two main components of a PWM signal that define its behaviour.

1. Duty cycle defines the amount of time the signal is in **ON** state as a percentage of the total time taken to complete one cycle. 100% duty cycle corresponds to fully high (**ON**) state setting the signal to Vcc, whereas 0% duty cycle is same as grounding the signal.
2. Frequency determines how fast the PWM completes a cycle i.e. how fast the PWM switches between high and low states.

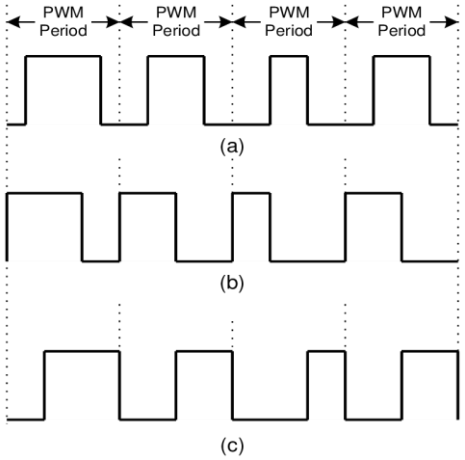
PWM types

<u>Symmetric</u>

1. Pulses of the PWM signal are always symmetric with respect to the centre of each PWM period. Refer the figure (a) below.

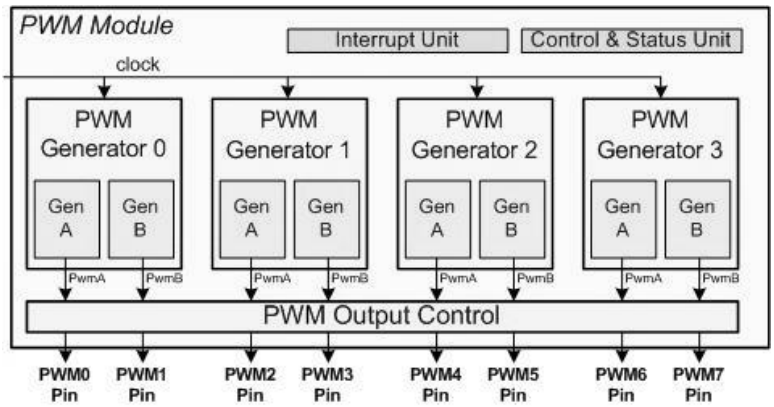
<u>Assymetric</u>

1. Pulses of the PWM signal always have the same side aligned with one end of each PWM period so there are Left aligned and Right aligned PWM signals in this case. Refer fig (b) and (c). This type of PWM can be used for stepper motors.



PWM Modules in TM4C123GH6PM MCU

The TM4C123GH6PM MCU contains two PWM modules, PWM0 and PWM1. Each module has **4** generator blocks and a control block. Each generator block creates **2** PWM output signals. This explains that a total of **16** PWM signals can be generated by these two modules. The control block determines the polarity of the PWM signals.



A PWM modules has –

1. 4 Generators
2. A counter (timer) (16 bit)
3. 2 compare registers **CMPA** and **CMPB**
4. 8 PWM output pins

The **counter** is programmed to count-up or count-up/down. As the counter counts, it checks whether its value matches with the value in the compare register, is set to 0 or is reloaded. Accordingly the output PWM is toggled, driven LOW or driven HIGH.

When the value of the counter, updated subsequently in the LOAD register matches either of the compare register values, a single-clock-cycle-width HIGH output is generated. If the value in either of the compare register does not match the HIGH signal is not generated.

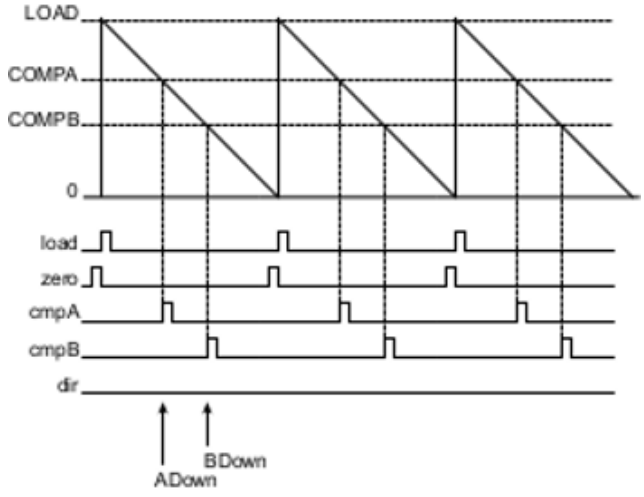


Fig a. PWM count-down mode

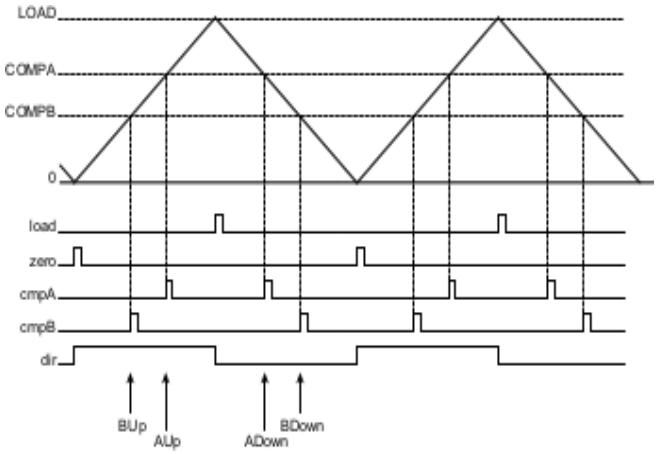
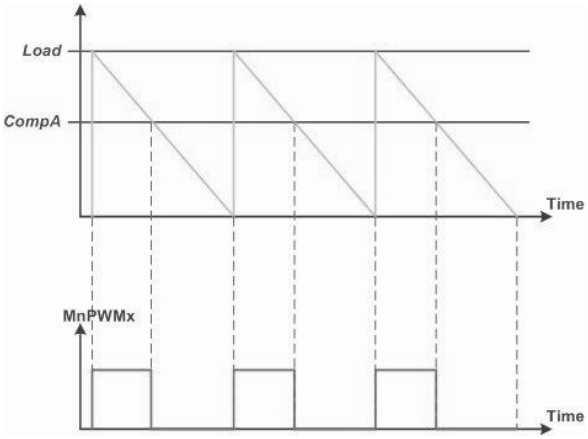


Fig b. PWM count-up/down mode

Programming PWM

The GPIO pins can be configured with the alternate function select option. Once the specific ports are set as PWM output pins, the Port Control registers need to be assigned the bit field encoding values in order for the GPIO pins to be used as PWM signal generating ports. Depending on the values from the MPU6050 sensor and the PID controller block correction values are evaluated and that correction is assigned to the duty values. We have considered a linear relationship between duty and correction, and accordingly the Kp, Kd, and Ki have been tuned as per Ziegler-Nichols method. To generate a periodic square wave using down counting mode, the PWMxGENx register is configured so that the output is driven high when the counter is reloaded and the output is driven low when the counter matches a comparator register while counting down.



The board supplies the output of 3.3V from each of its GPIO pins. As the input pins of the **L298N** motor driver require an input of 5V we make use of a level (logic) shifter

for this purpose. The following connections are present between the TM4C123GH6PM MCU and the L298N motor driver.

PA7 -> ENB, **PF0 -> ENA**

PE2 -> IN4, **PE1 -> IN1**

PE3 -> IN3, **PF1 -> IN2**

*Note: Here PA7 resembles pin 7 of Port A, PE2 resembles pin 2 of Port E and so on.

```
void pwm_init(void)
{
    SYSCTL_RCGC2_R |= 0x00000031;
    GPIO_PORTE_DIR_R |= 0x0E;
    GPIO_PORTE_DEN_R |= 0x0E;
    GPIO_PORTA_DIR_R |= 0x80;
    GPIO_PORTA_DEN_R |= 0x80;

    SYSCTL_RCGCPWM_R |= 2;
    SYSCTL_RCC_R &= ~0x00100000;
    GPIO_PORTF_LOCK_R = 0x4C4F434B; /* 2) unlock GPIO Port F */
    GPIO_PORTF_CR_R |= 0x03;
    GPIO_PORTF_DIR_R |= 0x03;
    GPIO_PORTF_DEN_R |= 0x03;
    GPIO_PORTF_AFSEL_R |= 0x01;
    GPIO_PORTF_PCTL_R &= ~0x0000000F;
    GPIO_PORTF_PCTL_R |= 0x00000005;

    GPIO_PORTA_AFSEL_R |= 0x80;
    GPIO_PORTA_PCTL_R &= ~0xF0000000;
    GPIO_PORTA_PCTL_R |= 0x50000000;

    PWM1_1_CTL_R = 0; /* stop counter */
    PWM1_2_CTL_R = 0; /* stop counter */

    PWM1_1_GENB_R = 0x0000008C;
    PWM1_2_GENA_R = 0x0000008C;
    PWM1_1_LOAD_R = 16000;
    PWM1_2_LOAD_R = 16000;
    PWM1_1_CTL_R = 1;
    PWM1_2_CTL_R = 1;

}
```

```
void set_dutycycle1(int i)
{
    i = 14000-i;
    PWM1_1_CMPA_R = i;
    PWM1_2_CMPA_R = i;  //different numbers to make both motors equal speed
}

void forward(void)
{
    PWM1_ENABLE_R = 0x18;
    GPIO_PORTE_DATA_R |= 0x4;
    GPIO_PORTE_DATA_R &= ~0x0A;
    GPIO_PORTF_DATA_R |= 0x2;
}
void backward(void)
{
    PWM1_ENABLE_R = 0x18;
    GPIO_PORTE_DATA_R |= 0xA;
    GPIO_PORTE_DATA_R &= ~0x04;
    GPIO_PORTF_DATA_R &= ~0x2;

}
void stop(void)
{
    PWM1_ENABLE_R = 0x00;
    GPIO_PORTE_DATA_R &= ~0x0E;
    GPIO_PORTF_DATA_R &= ~0x02;

}
```

CONCLUSION AND FUTURE SCOPE

Conclusion

The values of the sensor (MPU6050) have been rectified to almost idealism and ready to use with necessary offsets in the codes. The values are then fused to get the best estimate of the tilt angle using Complimentary Filter. A basic understanding of the Kalman Filter has been achieved by verification of tilt angle estimation on UART but it being computationally cumbersome isn't being used. The tilt angle estimated is PID tuned to direct the wheels to stabilize in upright position. The robot's physical design has been completed and the progress has been achieved to date as planned.

Future Scope

1. The Kalman Filter code is verified and is ready with necessary documentation for proper understanding. However, it is essential to debug the reason for delay (one reason might be because of matrix calculations) during computations in the Kalman Filter, and then successfully document it. Use of Kalman Filter is stressed as it seems to give better performance than the complementary filter.

2. The bot needs to be made a Line Following bot using IR array sensor or Line Scan Camera. The functionality of the code for line following and controlling the individual motors was checked and it was successful. However, we were unable to execute it using IR array sensor because the base layer of the bot was at a greater height that the maximal sensing range of the sensor. Interfacing a line scan camera can be an alternative to IR array sensor.

3. If the goal of line following can be achieved and the direction control for the wheels is possible by the bot itself, then a wireless remote controller can be an upgrade to be used with the bot.

4. Dynamic loading of the bot can be carried out, placing certain weight away from the center of the bot. This would change the center of mass of the system. Thus the bot can be made to react in appropriate ways under such conditions through Adaptive Tuning. This would require to take into consideration the state space model of the bot in order to predict the next state of the bot.

APPENDIX

Tiva™ TM4C123GH6PM

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	-
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	-
PA2	19	-	-	SSI0Clk	-	-	-	-	-	-	-	-	-
PA3	20	-	-	SSI0Fss	-	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0Rx	-	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0Tx	-	-	-	-	-	-	-	-	-
PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-	-
PA7	24	-	-	-	I2C1SDA	-	M1PWM3	-	-	-	-	-	-
PB0	45	USB0ID	U1Rx	-	-	-	-	-	T2CCP0	-	-	-	-

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PB1	46	USB0VBUS	U1Tx	-	-	-	-	-	T2CCP1	-	-	-	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-	-
PB3	48	-	-	-	I2C0SDA	-	-	-	T3CCP1	-	-	-	-
PB4	58	AIN10	-	SSI2Clk	-	M0PWM2	-	-	T1CCP0	CAN0Rx	-	-	-
PB5	57	AIN11	-	SSI2Fss	-	M0PWM3	-	-	T1CCP1	CAN0Tx	-	-	-
PB6	1	-	-	SSI2Rx	-	M0PWM0	-	-	T0CCP0	-	-	-	-
PB7	4	-	-	SSI2Tx	-	M0PWM1	-	-	T0CCP1	-	-	-	-
PC0	52	-	TCK SWCLK	-	-	-	-	-	T4CCP0	-	-	-	-
PC1	51	-	TMS SWDIO	-	-	-	-	-	T4CCP1	-	-	-	-
PC2	50	-	TDI	-	-	-	-	-	T5CCP0	-	-	-	-
PC3	49	-	TDO SWO	-	-	-	-	-	T5CCP1	-	-	-	-
PC4	16	C1-	U4Rx	U1Rx	-	M0PWM6	-	IDX1	WT0CCP0	U1RTS	-	-	-
PC5	15	C1+	U4Tx	U1Tx	-	M0PWM7	-	PhA1	WT0CCP1	U1CTS	-	-	-
PC6	14	C0+	U3Rx	-	-	-	-	PhB1	WT1CCP0	USB0EPEN	-	-	-
PC7	13	C0-	U3Tx	-	-	-	-	-	WT1CCP1	USB0PFLT	-	-	-
PD0	61	AIN7	SSI3Clk	SSI1Clk	I2C3SCL	M0PWM6	M1PWM0	-	WT2CCP0	-	-	-	-
PD1	62	AIN6	SSI3Fss	SSI1Fss	I2C3SDA	M0PWM7	M1PWM1	-	WT2CCP1	-	-	-	-
PD2	63	AIN5	SSI3Rx	SSI1Rx	-	M0FAULT0	-	-	WT3CCP0	USB0EPEN	-	-	-
PD3	64	AIN4	SSI3Tx	SSI1Tx	-	-	-	IDX0	WT3CCP1	USB0PFLT	-	-	-
PD4	43	USB0DM	U6Rx	-	-	-	-	-	WT4CCP0	-	-	-	-
PD5	44	USB0DP	U6Tx	-	-	-	-	-	WT4CCP1	-	-	-	-
PD6	53	-	U2Rx	-	-	M0FAULT0	-	PhA0	WT5CCP0	-	-	-	-
PD7	10	-	U2Tx	-	-	-	-	PhB0	WT5CCP1	NMI	-	-	-
PE0	9	AIN3	U7Rx	-	-	-	-	-	-	-	-	-	-
PE1	8	AIN2	U7Tx	-	-	-	-	-	-	-	-	-	-
PE2	7	AIN1	-	-	-	-	-	-	-	-	-	-	-
PE3	6	AIN0	-	-	-	-	-	-	-	-	-	-	-
PE4	59	AIN9	U5Rx	-	I2C2SCL	M0PWM4	M1PWM2	-	-	CAN0Rx	-	-	-
PE5	60	AIN8	U5Tx	-	I2C2SDA	M0PWM5	M1PWM3	-	-	CAN0Tx	-	-	-
PF0	28	-	U1RTS	SSI1Rx	CAN0Rx	-	M1PWM4	PhA0	T0CCP0	NMI	C0o	-	-
PF1	29	-	U1CTS	SSI1Tx	-	-	M1PWM5	PhB0	T0CCP1	-	C1o	TRD1	-
PF2	30	-	-	SSI1Clk	-	M0FAULT0	M1PWM6	-	T1CCP0	-	-	TRD0	-
PF3	31	-	-	SSI1Fss	CAN0Tx	-	M1PWM7	-	T1CCP1	-	-	TRCLK	-
PF4	5	-	-	-	-	-	M1FAULT0	IDX0	T2CCP0	USB0EPEN	-	-	-

MPU6050

The following link is for the datasheet of the sensor which contains the register maps of the sensor. Please refer the link to understand the interfacing of the sensor with the MCU.

https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf