PID Control System Design for IMU Sensors

Debuggin tool for MPU 6050

Hemaya S
Computer Science and Engineering
Adichunchanagiri Institute of Technology,
Jyothinagara, Chikkamagalur – 577102, Karnataka India

Sinchana Acharya
Computer Science and Engineering
Adichunchanagiri Institute of Technology,
Jyothinagara, Chikkamagalur – 577102, Karnataka India

***Abstract*—Proportional-integral-derivative(PID) Controllers are basically used in ICS – Industrial Control Systems due to the reduced number of parameters. These parameters are tunable. This paper concentrates on the work done on the PID control system specially for IMU (Inertial Measurement Unit) type of sensors. We focus on MPU 6050 – a gyro accelerometer sensor which is widely used to study and understand the relative position of any object on earth. Currently, the existing system, offers less way to debug the sensor values through approximation which is not accurate at times. In this paper, we are focused in developing a program along with some headers which can bring out the PID values, along with other parameters essential for debugging the relative position of the sensor at any given time.**

Keywords— Robust PID Control, Arduino, Header Files, PID Values, MPU 6050

I. INTRODUCTION

PID controllers are the most popular controller system used in industries, robust PID controller gives more robust output than classical PID controller. Their parameters are tuned according to the instability and delay induced by networked system and creates a safety margin in terms of phase and gain margins. They provide control signals that are proportional to the error between the reference signal and the actual output (proportional action), to the integral of the error (integral action), and to the derivative of the error (derivative action), namely

$$u(t) = K_p\left[e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{d}{dt}e(t)\right]$$

where ( ) and ( ) denote the control and the error signals, respectively, and , and are the parameters to be tuned. The corresponding transfer function is given as

$$K(s) = K_p\left(1 + \frac{1}{T_i S} + T_d S\right)$$

The main features of PID controllers are the capacity to eliminate steady-state error of the response to a step reference signal (because of integral action) and the ability to anticipate output changes (when derivative action is employed).

The InvenSense MPU-6050 [1] sensor contains a MEMS [2] accelerometer and a MEMS gyro. It is very accurate. It contains 16-bits analog to digital conversion hardware for each channel. It also captures the x, y, and z channel at the same time for any given axis. The sensor uses the I2C [3] (Inter-Integrated Circuit) bus for communication.

Arduino / Genuino Uno [4] is a microcontroller board based on the ATmega328P. It has a set of 14 digital input/output pins (of which 6 can be used as PWM outputs) and 6 analog inputs. A 16 MHz quartz crystal acts as the heart of the Microcontroller, a USB connection for communicating with computers, a power jack, an ICSP header and a reset button. The way it works is simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery and start programming.

II. LITERATURE SURVEY

A thorough study was carried out to find, how basically a PID Controllers are designed programmatically and tuned. During, this study we came across several huddles, such as there was no definitive program to determine the exact position of the sensor at any given time. These sensors are basically used in
Aeronautics, Flying equipment's, and to stabilize objects on any form of two legs or wheels. In order to achieve this, we need the Yaw, Pitch and Roll values of the sensor. These are calculated from the sensor using Program. But they provide only the final values of YPR (Yaw, Pitch, Roll) but not the initial angles of impact. With the final YPR values, it is impossible to determine the angle of impact and also find the stable , and

A. Why MPU 6050

For any IMU (Inertial Measurement Unit), to determine whether the object is falling or accelerating towards a direction, we need a Gyroscope and Accelerometer. MPU 6050 Is a combination of both of these and also has a 6 degree of freedom. Basically, 3 degree of freedom is quite enough for any starters, but we wanted to start from the intermediate state, so MPU 6050. Most importantly, it is a I2C device.

B. How is PID calculated

Arduino uses a basic robotic protocol called I2C to communicate with I2C devices. For this, they have something called a Wire Library [5]. This library is basically used to communicate with I2C / TWI devices. On the Arduino Uno boards with the R3 layout (1.0 pinout), the SDA (data line) and SCL (clock line) are on the pin headers close to the AREF pin. There are both 7- and 8-bit versions of I2C addresses. 7 bits identify the device, and the eighth bit determines if it's being written to or read from. The Wire library uses 7 bit addresses throughout. The addresses from 0 to 7 are not used because these are reserved therefore the first address that can be used is 8.

For simplicity purposes, we have chosen, Arduino Uno. This is the basic version, which has the capability to handle all types of IMU Sensors along with I2C communication.

I2C for Arduino was created by DSS Circuits [6] and another version by Rambo [7]. These were developed more than 5 years back, the program used to debug or calibrate the PID is provided by Brett Beauregard [8].

III. CURRENT SYSTEM DESIGN

Current program which include a PID Header and CPP files are designed to just display whether the device is alive or not. Whether the DMP is enabled or not. But, the program fails to provide the exact input values to the sensor, or , and

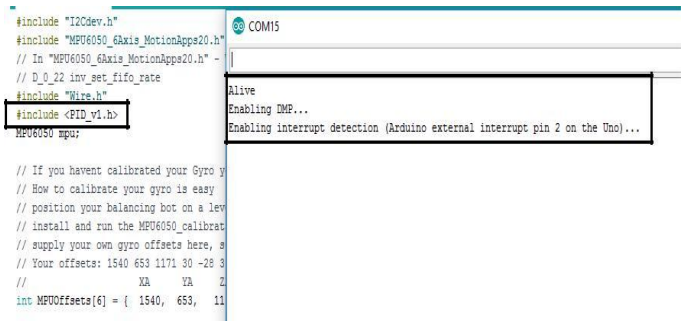

Fig. 1. PID_V1 Values for a MPU 6050

Let us look into the PID_v1 header along with its CPP file to see, what the headers are intended to do.

A. *PID_v1.h*

This header file [9] basically is used to calculate PID values but never display them. They use the .cpp files to get the input from the sensor, manipulate them using some functions and then derived those outputs which can be used by the controller for navigating the robot. This file, just declares all the functions which will be used in calculating the , and values. These functions include: PID, Compute, SetTunings, SetSampleTime, SetOutputLimits, SetControllerDirection and Initialize.

B. *PID_v1.cpp*

- This .cpp file [10] imports files from Arduino and WProgram to enable it to interact with the microcontroller and sensor.
- Function PID: A common constructor to give some predefined generic values, as the program starts with some unreliable defaults.
- Function Compute: This function will calculate a new PID value if needed else it won't. It returns true if the output is computed else false.
- Function SetTunings: The basic use of this part is to make changes to the controller's dynamic performance to be adjusted. It is either called manually or from the constructor.
- Function SetSampleTime: This function sets a time frame at which the calculation has to performed every time.
- Function SetOutputLimits: This function is used more often than SetInputLimits. This is done basically because the input is of the range 0 – 1023 which is fixed. But, the output may vary depending upon the programs calculation and usually it is not so predictable. Hence, to reduce the output given by the program within a given set of ranges i.e., 0 – 125.
- Function SetMode: This function allows the controller mode to be set to manual (0) or automatic(non-0).
- Function SetControllerDirection: This function will make the PID be connected to a DIRECT acting process or a REVERSE acting process.
- Function Status: These functions return the status to the controller.

C. *Limitations*

- No Input value is seen on the output screen.
- Three key terms, which were calculated, are never displayed but only returned to the program. This makes it difficult and more program code has to be present in the Arduino to debug it, which in turn takes more memory in the controller, thus decreasing the performance.
- It makes it hard for basic beginners, to debug and develop a program without knowing the , and values.
- Input values to the sensor and output values to the sensor are never seen on screen.

D. *Memory usage*

This sketch uses 17564 bytes (54%) of program storage space. Maximum is 32256 bytes. Global variables use 909 bytes (44%) of dynamic memory, leaving 1139 bytes for local variables. Maximum is 2048 bytes.

These limitation, has been the inspirational and driving factor for me to develop a new header and class file that will provide all these values and also better help as a debugger program

IV. PROPOSED SYSTEM – PID_V2.H & PID_V2.CPP

After analyzing detailly on how PID works on IMU type of sensors, we developed a program, that without interrupting the system, would also capture the values of , and and
display them along with other parameters on the serial monitor. This will allow us to determine at what exact position was the sensor last seen, what is the input of the sensor, what is the values calculated and what will be the output. The challenge would be to achieve this, without increasing the time or without creating any additional interrupts that may slow down the process.

A. *PID_v2.h*

The header file is similar to the one of the PID_v1 but it has some additional components. It has a DEBUG with the following parameters – Name, Variable, Spaces, Precision,

TempStringName which in turn are retrieved using the .cpp file and displayed

B. PID_v2.cpp

- This .cpp file imports files from Arduino and WProgram to enable it to interact with the microcontroller and sensor. The .cpp file also includes all the parameter of the PID_v1.cpp with 2 new additions. 1 a new Function and 1 Definition. The Definition carries out the part of debugging, getting values and displaying them on screen.
- Definition DEBUG: It is defined to capture values on the stream between calculation and display them for every 100 milliseconds.
- Function PrimeIntegral: This function forces a value to the output to prime for restart in case of any stalls.

C. Advantage over PID_v1.h

1) The following table clearly shows the value that will be seen on the screen during debug or output mode

TABLE I. OUTPUT VALUES DISPLAYED

S. No	Serial Monitor							
	Input	*SetPoint*	*DeltaTus*	*DeltaTs*	*PTerm*	*ITerm*	*DTerm*	*Output*
1								

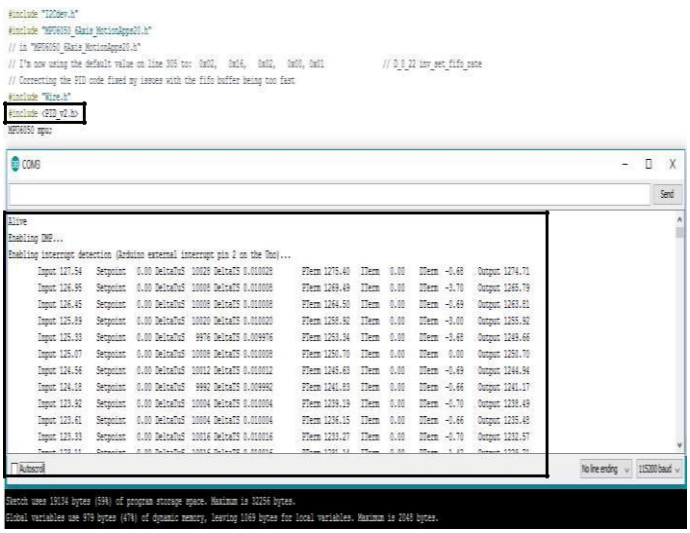

Fig. 2. PID_V2 Values for a MPU 6050

Figure 2 clearly shows the use of PID_v2.h header file and the output with all the parameters as displayed in the table.

2) Lesser or same time equivalent in terms of program execution and results display without any lag

D. Memory

Sketch uses 19134 bytes (59%) of program storage space. Maximum is 32259 bytes. Global variables use 979 bytes (47%) of dynamic memory, leaving 1069 bytes for local variables. Maximum is 2048 bytes.

V. RESULTS

A detailed analysis was carried out on both the programs.

Varying from Big Omega (Ω) to Big O (O) and Small O (O) to Small Omega (ω). The following are the finding between the both headers

A. Program Execution

When the program was executed, the PID_v2.h seems to have an advantage over the PID_v1.h. This is due to the fact that since the program is reduced in lines has a better execution rate.

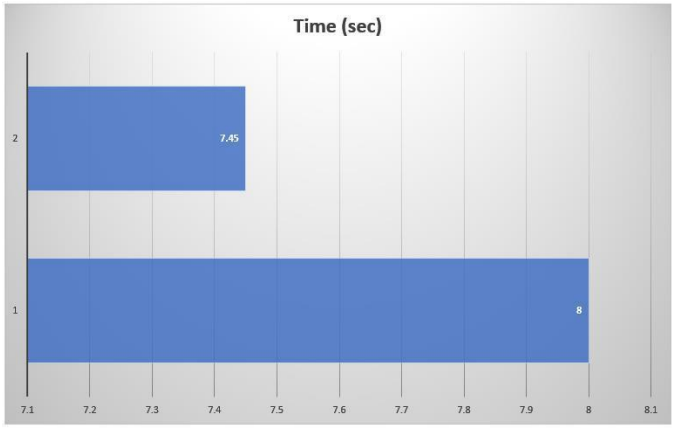

Fig. 3. Program Execution time between PID_v1 and PID_v2

B. Memory Consumption

The memory consumed by PID_v2 is higher because the initial memory load for the program is higher for the header files than the PID_v1. This is due to the fact that extra parameters are added on to the header files

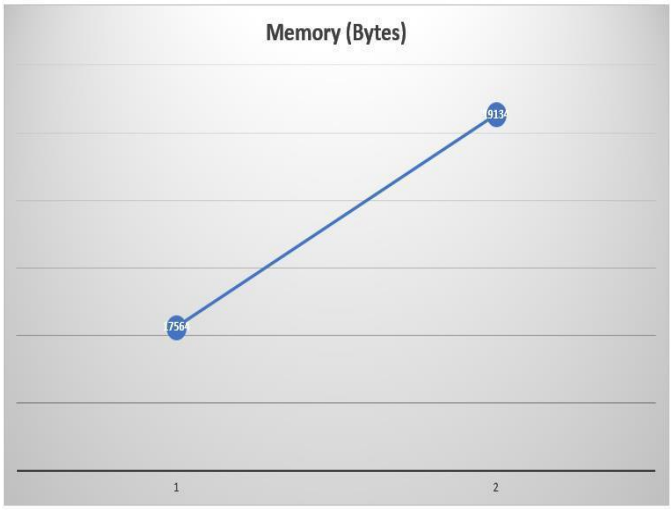

Fig. 4. Memory consumbed between PID_v1 and PID_v2

But, this one has an added advantage. Instead of PID_v1 which has to loop all variables and thus consuming the same amount of memory every loop, but PID_v2 is a onetime load and the rest of the cycle doesn't require the initial load memory.

C. Global Variables

Since all the variables are consumed by global parameters, the PID_v2 global values are slightly greater than the PID_v1.
Usually this doesn't have any impact on as these are loaded at the beginning of the program. This may induce some time lag, but those are negligible.

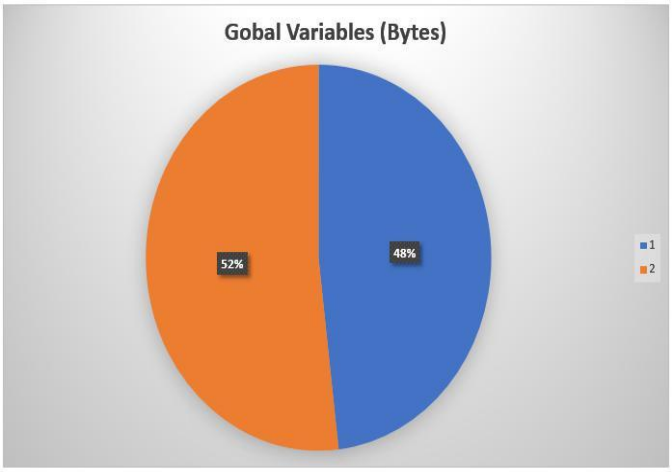

Fig. 5. Global Variable Memory Consumption between PID_v1 and PID_v2

D. Local Variables

The memory allocated for local memory is decreased. As the global variables are having higher values on PID_v2, this will directly impact the memory allocated for local variables.

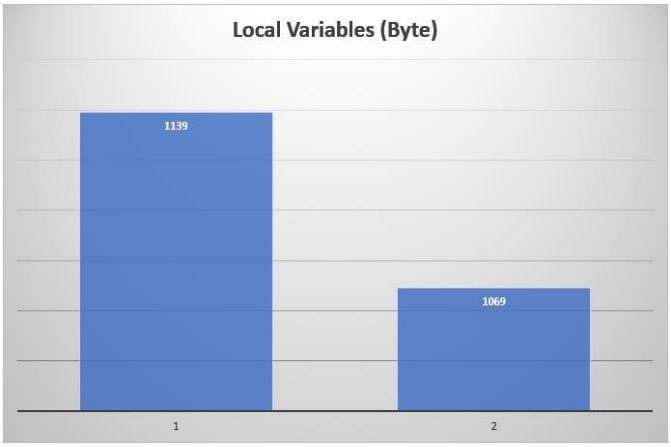

Fig. 6. Local Variable Memory Consumption between PID_v1 and PID_v2

VI. CONCLUSION

Upon creating the new header file and class file, we were able to fill in the bridge between the Sensor values and also the Display. The main aim of this paper was to display the sensor reading of any IMU Sensor, to help the students calibrate the sensor reading to accuracy and also use it for debugging purposes. A future study will be made to improvise this header file and class file. So, that less memory consumption and greater efficient execution of programs on robots can be designed. This will impact on lesser time for execution, lesser memory for program, lesser global variables and more local variable memory for the robot to consume. Optimization of Big Omega (Ω), Big O (O), Small O (O) and Small Omega (ω) should be fine – tuned, so that it can be used under all scenarios and situations.

ACKNOWLEDGMENT

I would like to thank my school teacher Mrs. L. Rajeshwari, who laid the stepping stone for my career towards computer programming. I would also like to thank my college professors, who has been very kind and helpful in this regards, throughout the project.

REFERENCES

[1] http://playground.arduino.cc/Main/MPU-6050, https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/
[2] https://www.mems-exchange.org/MEMS/what-is.html
[3] https://en.wikipedia.org/wiki/I%C2%B2C
[4] https://www.arduino.cc/en/main/arduinoBoardUno
[5] https://www.arduino.cc/en/Reference/Wire
[6] http://dsscircuits.com/index.php/articles/66-arduino-i2c-master-library
[7] https://github.com/rambo/I2C
[8] https://github.com/br3ttb/Arduino-PID-Library
[9] https://github.com/br3ttb/Arduino-PID-Library
[10] https://create.arduino.cc/projecthub/twob/self-balancing-robot-using-blubug-8894c6