Delivery Drone: Frame and Control Software

EE 496 Report
Fall 2014

Department of Electrical Engineering
University of Hawaii

Dee Mosher

December 18, 2014

Faculty Advisor: Professor David Garmire

1. Introduction.

This project is a continuation of a project started last semester. This paper will focus on the progress of the quadcopter's frame, a report about the flight software, including a summary about PID loops, and details into the current RC control system and data visualization methods. All of this will be concluded with research and design considerations.

2. Background

Quadcopters are a type of multirotor helicopter that utilizes four rotors for lift, stabilization and precisely controlled movement. These four rotors eliminate the need for the mechanically complex pitch-altering rotors found in traditional helicopters and instead relays on changing the speed of the different rotors to achieve variable movement in three dimensional space. Two of the four propellers rotate clockwise while the other two rotate in a counterclockwise direction. By varying the speed of the different propellors, four degrees of movement can be achieved including side to side, forward and backward (by manipulating the pitch by setting two rotors on one side to be faster than the other), up and down (by increasing the speed of all rotors), and horizontal rotation (by making the two propellors that are rotating the same direction faster than the other two, thus increasing the torque in a certain direction).

In recent years, multicopters, primarily quadcopters, have started to gain a large foothold in the commercial Unmanned Aerial Vehicle (UAV) market. Due to their stability, size, hovering capabilities and rechargeable batteries and the advent of small, affordable, high definition cameras, they are idea for inexpensive low-altitude aerial photography. Many real estate agencies have started to use them to inexpensively take photographs of properties, while athletes, especially those engaged in "extreme" sports, have used them for taking above-the-action footage of games and stunts.

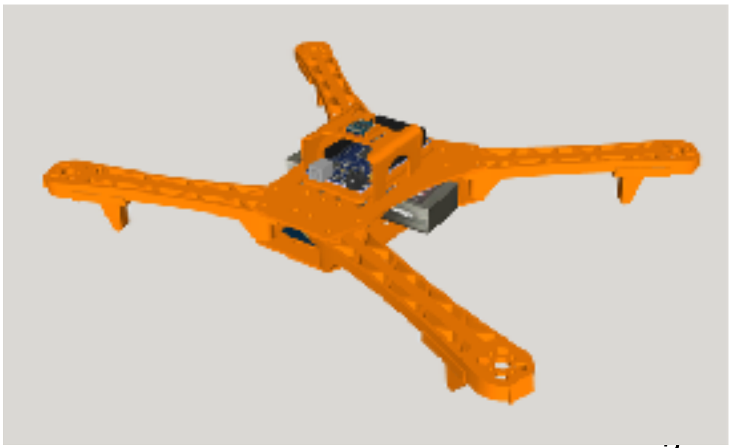
DHL's delivery quadcopter

Recently, several companies have started looking into using quadcopters and other types of UAVs for the delivery of small packages. Amazon PrimeAir is probably the most publicly known multicopter delivery service, but DHL has also taken the plunge using a quadcopter to deliver medications and other urgent items to the German island of Juist.

3. Project Objectives and Criteria

Our final objective with this project is to fix a problem current delivery quadcopters face: horizontal movement is not efficient. We plan on solving this by utilizing airfoils like an airplane.

Before attempting to build an airfoil utilizing flight system, one which would be incredibly difficult to realize, especially with little to no aeronautical engineering experience, we decided to first work on a


Render of previous frame design

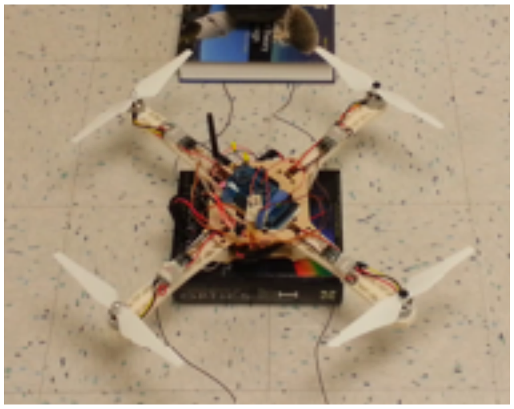
traditional quadcopter. By doing this, we would also build a quadcopter system that could be used for other projects. Once we finish building this portion, we would then move onto solving the inefficiencies found in horizontal flight.

4. Quadcopter Systems

There are several parts, both hardware and software, that make up the current incarnation of our quadcopter that this portion of the paper will cover. In hardware, we have the frame and the control electronics and sensors, which are made up of the Electronic Speed controllers (ESC), the accelerometer, the gyroscope and the Arduino. The flight software consists of a Proportional-Integral-Derivative (PID) controller, a Kalman filter and the controller interface, all of which can be debugged using a data visualization and logging program.

4.1.Frame

Currently, we are using a symmetrical quadcopter frame. Last semester, our group used what is known as an asymmetrical frame. After much research and redesigning of the electrical system, we switched to a symmetrical design. Asymmetrical frames are primarily used to keep propellers out of view of a first person POV camera system, where the camera is affixed directly to the front of the frame, something we originally considered having. These systems are slightly more difficult to program, as the pitch and roll PID values would be different values and would need to be tuned independently. After removing the need for a forward facing camera as a necessity, we realized that this frame configuration was unnecessary. The only camera system we planned on adding was a bottom mounted gimbal system, which would not noticeably benefit from the clearance offered by an asymmetrical design.


Current quadcopter frame during a hover test

The frame from last semester was completely 3D printed. This semester, we utilized 3D printing for the arms and laser cut plywood for the center. The 3D printed arms of the quadcopter were made from ABS plastic printed from the uPrint SE Plus 3D printer. Both ABS and PLA plastics were printed as experiments.

4.2.Arduino and ESCs

Last semester, we started with the Arduino Mega, one of the more powerful members of the well known Arduino microcontroller board family which uses the ATmega1280 microcontroller. At the beginning of this semester, we ordered an Arduino Due, a board with a faster clock speed (84 MHz vs 16MHz) based on the 32 bit ARM core AT91SAM3X8E. We had to switch back to the Mega because the output pins of the Due had a operating voltage of only 3.3v, not the 5v that was required to control the ESCs.

The microcontroller was programmed using the Ardunio Integrated Development Environment (IDE). A few members had to use a separate microcontrollers for programming specific portions. During these time, we used an Arduino Uno.

The ESCs allow the Arduino to control how much power each motor gets without having large amounts of current going through the Arduino. We used the Castle Creations QuadPack 35 ESCs, four 35 amp Multi-Rotor ESCs, all of which are reprogrammable using a Castle Link USB adapter and Castle Creations' Castle Link. One of these ESCs contains a Battery Eliminator Circuit (BEC) which supplies power to the rest of the ESCs. Each ESC operates at 5V. These ESCs can be programmed to customize values for low-voltage cutoff, prop brake strength and governor mode gain.

4.3.R/C Controller and Receiver

When we began, we controlled the Arduino through codes entered into the Arduino IDE serial monitor. By typing "A" we armed the ESCs, "O" set the throttle to a set amount and "F" shut off the motors and disarmed the ESCs. This worked for simple testing, but we soon realized that we needed an easier, more accurate, more responsive and more intuitive way to control the quadcopter. We realized that the simplest solution to this problem was by using a controller.

RadioLink T6EHP-E Controller

Our controller utilizes the T6EHP-E OEM Digital 6 channel Proportional R/C System for control and communication. The T6EHP-E system was designed, like most RC controllers, to be used to control servos using a PWM signal with a period of about 20 milliseconds. We are able to read this signal by using Arduino's pulseIn(pin, value, timeout) function, which we set to wait for a "HIGH" signal on a pin to start a timer and then output the duration of the "HIGH" signal in microseconds. To ensure that this function wouldn't hang for an unacceptable amount of time when it didn't register a signal. Currently, only throttle is implemented, but code has been written that would allow all the inputs on the controller to be utilized. forward/backward, up/down, left/right, pitch, yaw, and roll. Throttle was important to implement for PID value tuning.

We found that the T6EHP-E controller has built in functions that changes the behavior of the receiver. This initially confused me, as I thought the throttle changed in a purely linear manner, especially when the settings were such that what should have been full throttle was set to be lower than what was expected to be about 75% throttle. It turns out that some RC pilots change the throttle curve depending on their preferences for different flight behaviors and equipment. It would make sense that these pitch and throttle curves can be changed. The main three points are the idle point, which is the lowest setting, the center stick point, intended to be the throttle that allows the craft to hover in place, and the max rpm point, the highest setting. We also found that some of the control sticks changed the same channel, which would make sense as these controllers tend to be used for both helicopters and airplanes, both of which have differing behaviors that could be mapped to a similar controller. For the most part, we used factory settings and decided to do any changes to the controls beyond trim through software.

We utilized the Arduino Integrated Development Environment (IDE) to develop all the software on board the quadcopter, including the software used for reading the transmitter. unsigned int RCread(int pinin, int chanmin, int chanmax) is the function that measures the current throttle using digital capture. pinin refers to which pin it reads, chanmin refers to the predetermined minimum value of that channel, and chanmax refers to the predetermined maximum value of that channel.

RCread not only reads the value, but uses the map function to map the digital capture value (between 1755 at minimum throttle to 879 at max on the controller) to throttle values the ESCs would accept (between 150 at low speed and 254 at high). Going below 150 would disarm the ESCs, and values above 1755 would map to a value below 150, so the unsigned int cutoff(int rawin,int minthrot,int maxthrot) function that made sure no throttle values went above or below accepted ranges.

Each degree of movement requires its own input port on the board. The Arduino sequentially reads each port on the receiver. In our test situation, we only had one port being used; throttle. The worst case maximum time it would take to read all 6 channels is about 112.2 ms. This is fine for maybe one PWM input on a system that doesn't need immediate responses, as pulseIn() delays the program whenever it is called but may be problematic when used for controlling a quadcopter, where delays could mean the difference of hitting or missing an obstacle or becoming unbalanced, paying attention to how small delays add up is important. This approach is fairly rough, which was fine debugging our stabilization system and calibrating our PID values, but would most likely cause problems if we were to use the controller for full motion.

A solution I considered would be to use a low pass filter to effectively change the PWM to a fairly smooth analog value that could be read as a value through the analog pins. The Arduino Mega has 16 10-bit analog to digital converters that map input voltages between 0 and 5 volts into integer values between 0 and 1023. analogRead() also has a delay. According to the Arduino Language Reference, this delay is about 100 microseconds, meaning to read all six channels would take a little over .6 ms. While using a digital pin to read the receiver pins using the pulseIn function, you could only read about 53 integer values per second, while feeding the PWM signal into a low-pass filter that was then read by the analog pins would result in a read rate close to 10,000 times per second. In the end, I decided not to implement this solution as the OEM receiver was a temporary solution and will be replaced by an XBee.

4.4.PID Controller

The Proportional Integral Derivative (PID) controller is a control feedback mechanism that calculates error values as the difference between a measured valued being processed and a desired value. Error is minimized by adjusting the processes with a manipulated value given to it. The PID algorithm consists of three individual constant parameters:

- Proportional (P) – depends on the present error
- Integral (I) – the accumulation of the past errors
- Derivative (D) – the prediction of the future errors based on the current rate of change.

A sum of these values are then used to adjust the process via a control element. In the quadcopter, the dampener is the controller values from the user.

The PID values need to be calibrated. Improperly calibrated values will cause serious oscillations that will destabilize flight.

5. Data Collection and Analysis

The visualization and data collection software used the Processing IDE. We looked into Comma-separated values (CSV) to format the data we got from the sensors for our final report and for better. We took a simple Processing graphing sketch and modified it. The first modification was made was to make the code automatically connect to a specified connection based on a shortened version of the connection's name (currently set to USB). The original processing sketch only showed one signal; we had to modify it to show several by having the Arduino sequentially send the data along with an identifier through a serial connection. The current version can only show three graphs within Processing, but it was written with the intent of being able to graph more.

6. Future Work and Development

The final goal of this project is to create a VTOL quadcopter system that can take off and land vertically like a quadcopter, but use an airfoil for horizontal flight with the intention of delivering small packages. There are still many things that need to be addressed and built for this to happen.

6.1.XBee and Controller

In the future, we are planning on using an XBee for flight control using a USB controller. We originally used an XBee for control via the Arduino IDE serial monitor. Using the XBee would also allow for communication from the quadcopter back to the user. Currently, using the OEM receiver, we are only able to send data one way; from the controller to the quadcopter. It would also allow us to use custom antennas, such as the one recently developed by a fellow EE 496 project who we have been in close contact with.

The current control and communication system is fairly basic. It only affords us one-way communication to the quadcopter. While adequate enough for our debugging and cost effective, realistically, this system doesn't work well. We hope to replace our current system with a system that utilizes an Arduino Uno with two shields: a USB shield and a protoshield that the XBee can connect to. The USB shield would hook up to a USB controller, most likely a commercially available USB gaming controller (current decision is either Xbox ONE or 360). The UNO would read the game controller's current state and then send that information to the quadcopter via XBee. The UNO should also be able to read data coming into it. This would allow us to easily and wirelessly record data from the quadcopter.

Effectively, we would build our own integrated controller. If we were to integrate a camera onboard, we could also add in a small monitor.

6.2.FAA Regulations

Of great interest to our project is the on going debate about drones being used for delivery in the United States. Technically, our research would probably require a Special Airworthiness Certificate which, according to the FAA website would:

> preclude carrying people or property for compensation or hire, but do allow operations for research and development, flight and sales demonstrations and crew training.

Something else to note is the fact that public research universities can file for FAA permission in limited numbers, while private institutions can't. While certification is clearly a hassle and the FAA has butted heads with many organizations about their rules (including a recent case where 188 research universities filed against the FAA) the FAA certification restrictions may work in our favor.

The Federal Aviation Administration has stated that it "plans to allow commercial UAS use once it has drawn up proper regulations for the aircraft. The deadline for those regulations is some time in 2015." Until the FAA proposes clear guidelines the use of UAV or drones for construction and industry use is confusing in what regulations apply. One of the FAA's grey area rulings appears to be the commercial versus hobbyist use of drones. This past year, the FAA fined Raphael Pirker, a $10,000 fine for flying a model airplane for hire in 2011. As the article goes on to explain: "With the tremendous interest and growth of a low-priced, readily available hobbyist drone flying public it seems imperative that common sense ruling on drone use are FAA established."

A question I ask myself is if we were to create an FAA certified platform that can be modified for other UH research, would others want to go through all the same red tape and problems instead of using an already platform? Would projects that use an already approved system need to get their own license? While FFA compliance is an issue to monitor, we are not currently doing outdoor flight tests. Until we get to that point, we will most likely not need to worry about breaking FFA regulations.

6.3.Future Frame Design

As stated earlier, due to problems with the Makerbot Replicator 2X, we sourced the majority of our parts from Punahou's uPrint SE Plus. Since building our frame, we have obtained a new 3D printer called the Printrbot Metal. The Printrbot has only one extruder and it extrudes a different kind of plastic called PLA, which has different properties to the ABS we are current using.

Both PLA (Poly(lactic acid)) and ABS (Acrylonitrile butadiene styrene) have the properties of becoming soft and malleable when heated and solid once cooled. Each plastic has the potential of being reused as the process of heating and reforming can be repeated again and again. In the 3D printer the plastic extrusion from a heated head melts a supply spool of filament, building layer on layer in a heated chamber. The heat needs to be carefully controlled to allow layers to fuse together but not misshape. Most 3D printers, including the uPrint SE Plus, have a second support material that is deposited in unison to assist in holding soft plastic in place. The Printrbot doesn't have support Material. Any supports are printed with the same material and need to be manually removed, usually with a knife. Because of this, 3D objects made on these printers should have only a few overhangs.

A number of pros and cons are associated with each plastic. Both need to be carefully stored with attention to moisture. When ABS is moisture laden, it will tend to bubble and spurt from the tip of the heated printing nozzle, reducing the visual look of the part, the print accuracy and strength, and clog the nozzle. Spools of ABS can be dried before printing using a source of hot air or a dehumidifier.

PLA may also bubble or spurting at the printer nozzle when moist. Discoloration and inaccuracy may occur when. This is because PLA can react with water at high temperatures, causing the plastic to undergo a chemical process called depolymerization. Drying the spools of PLA can also be successful, but may cause change in extrusion temperature and plastic quality.

Generally speaking, the printing quality of ABS and PLA are very comparable. ABS adhering to the 3D printer platform is often the most important point of potential print failure. ABS will tend to curl upwards of the surface of the 3D Printer's print bed. ABS will often slightly round at a sharp corner which is an issue for prints that contain objects such as gear. The hot plastic smell of ABS comes from its petroleum based origin. PLA is much less inclined to warping when compared to ABS. For this reason it is possible to use a printer to print without a heated bed, but the plastic can still curl up slightly on large PLA parts with out heating the buildplate. PLA has larger phase-change when heated and becomes much more liquid than ABS. If printer cooling is in use, much sharper details can be achieved in sharp corners without the risk of cracking or warp. The Printrbot allows this through the use of a directed fan near the extruder. The property of PLA being less viscous can also create stronger binds between layers producing greater strength in printed part.

ABS has the advantage of machinability, along with a higher temperature resistance. While you can drill holes in ABS with ease, doing so with PLA will cause the piece to shatter. If we can, I think we should do some materials testing on the two plastics.

Another design change I have been considering is using more non-rapid prototyping materials. Aluminum rods are easy to source from a hardware store and cut to length. It is a fairly light, strong material and not too expensive. It may be possible to cut down on print times and, considering that smaller 3D Printed parts are less likely to have problems on a partially untuned 3D printer, misprints by replacing a large portion of the quadcopter's arms with aluminum. Of course, we will need to do some weight calculations before doing so.

6.4.Package Delivery

There are many design considerations that will need to be addressed when we start looking into delivering packages. As I did some brainstorming on the frame design for the delivery quadcopter, I realized there may be a few issues with lifting flat packages with a large area. Attaching a flat parcel to the bottom of the quadcopter would cause thrust issues if the bottoms of the rotors are covered. I'm wondering if a system could be developed that could partially incorporates the package structure into an extendable frame and whether it would be worth the trouble, thus lowering the amount of drag. At the very least, setting up the system to allow packages larger than the central "hub" is an interesting idea that needs more thought.

Variable weight is another problem; currently, we need to refine our PID values using a test rig, however, the PID values will change with a change in weight. I feel we need to investigate a way to tune the PID values versus mass. PID values could also change depending on how susceptible packages are to vibrations.

Originally, our team decided our final design utilize what is known as a a tilt rotor VTOL (Vertical Take Off and Landing) system, where the front and back rotors are fixed to the body while the lateral rotors would tilt forward for forward thrust. The tilt rotor system is fairly complicated in terms of balance and control, especially when tilting the rotors forward. Recently, we noticed GoogleX's Project Wing, which uses what is known as a tailsitter configuration. Tailsitters are VTOLs that tilt the entire vehicle's body, flaps and all, to transition from vertical to horizontal flight. Tailsitters were first explored in World War II, but were scrapped due to pilot visibility, an issue that does not apply to UAVs.

A possible system that could be useful would be swarm robots with each rotor being its own robot. These would be more complicated, but affords more package scalability. Using short range mesh communication, each swarm bot would attach to a part of the package and then release to deliver it. It could also help with irregularly shaped objects whose center of mass; rotorbots could be attached in an easier to control manner around the center of mass. A system like this may be difficult to automate in terms of loading packages.

Lastly, developing a blackbox system may be advantageous. It could be avoided, however, by storing information on the cloud, which would avoid adding extra weight to the drone.

7. Design Methodology

For code, we employed a top-down approach where we split up the work into parts and had each team member work on a separate portion. At the end, we stitched all the code back together.

8. Conclusion

While we still have a lot of work ahead of us, we were successful in redesigning the frame to utilize more expedient and cost effective manufacture techniques, built a working stabilization system that just requires some calibrations and tuning before it becomes functional.

References

"Arduino Mega" [Online document] [Online document] (2014 Winter). [2014 Dec 12] Available at HTTP: arduino.cc/en/Main/arduinoBoardMega
"Arduino Due" [Online document] [Online document] (2014 Winter). [2014 Dec 20] Available at HTTP: arduino.cc/en/Main/ArduinoBoardDue
"Support" [Online document] (2014 Sum). [2014 Dec 19] Available at HTTP: http://www.futaba-rc.com/faq/faq-7c-q756.html
"Quadpack Multirotor" [Online document] (2014 Sum). [2014 Dec 19] Available at HTTP: http://www.castlecreations.com/products/quadpack.html
"Introducing Prime Air" [Online document] (2014 Sum). [2014 Dec 20] Available at HTTP: http://www.amazon.com/b?node=8037720011
W. Grayson, "Eyes in the Sky: How Drones and UAVs Are Already Affecting Construction Jobsites" [Online document] (2014 Winter). [2014 Dec 22] Available at HTTP: http://www.equipmentworld.com/drones/

Source Code

```
//MPU_6050.ino
#include <Kalman.h>
#include <math.h>
#include <PIDCont.h>
#include <Wire.h>

#include "IMU_6050.h"

// Global definitions
#define ROLL_PID_KP    0.0007
#define ROLL_PID_KI    0.000007
#define ROLL_PID_KD    0.07
#define ROLL_PID_MIN  -52.0
#define ROLL_PID_MAX   52.0

#define PITCH_PID_KP   0.0007
#define PITCH_PID_KI   0.000007
#define PITCH_PID_KD   0.07
#define PITCH_PID_MIN -52.0
#define PITCH_PID_MAX  52.0

#define YAW_PID_KP     0.0007
#define YAW_PID_KI     0000007
#define YAW_PID_KD     0.07
#define YAW_PID_MIN   -52.0
#define YAW_PID_MAX    52.0

#define ANGLEX_KP      5.0
#define ANGLEX_KI      0.02
#define ANGLEX_KD     -0.015
#define ANGLEX_MIN    -104.0
#define ANGLEX_MAX     104.0

#define ANGLEY_KP      5.0
#define ANGLEY_KI      0.02
#define ANGLEY_KD     -0.015
#define ANGLEY_MIN    -104.0
#define ANGLEY_MAX     104.0

// Global variables
int event_armed = 0;
int event1 = 0;
int event2 = 0;
int event3 = 0;
const int ledPin = 13;
int incomingByte;
int temp;
float error_Y[5][2];
int count = 0;
PIDCont PIDroll, PIDpitch, PIDyaw, PIDangleX, PIDangleY;
int setX = 0;
int setY = 0;
int setZ = 0;
float gx_aver = 0;
float gy_aver = 0;
float gz_aver = 0;
float motor[4];  //current motor speed
float min_speed; //used to set speed modes between settings
float max_speed; //prevents random jumps or shut-offs
int throttle = 0;
Kalman kalmanX; // Create the Kalman instances
Kalman kalmanY;
float base_x_accel;
float base_y_accel;
float base_z_accel;
float base_x_gyro;
float base_y_gyro;
float base_z_gyro;
unsigned long last_read_time;
float last_x_angle;  // These are the filtered angles
float last_y_angle;
float last_z_angle;
float last_gyro_x_angle;  // Store the gyro angles to compare drift
float last_gyro_y_angle;
float last_gyro_z_angle;
float last_x_kalangle;  //Kalman angle
float last_y_kalangle;
float last_z_kalangle;
typedef union accel_t_gyro_union
{
  struct
  {
    uint8_t x_accel_h;
    uint8_t x_accel_l;
    uint8_t y_accel_h;
    uint8_t y_accel_l;
    uint8_t z_accel_h;
    uint8_t z_accel_l;
    uint8_t t_h;
    uint8_t t_l;
    uint8_t x_gyro_h;
    uint8_t x_gyro_l;
    uint8_t y_gyro_h;
    uint8_t y_gyro_l;
    uint8_t z_gyro_h;
    uint8_t z_gyro_l;
  } reg;
  struct
  {
    int x_accel;
    int y_accel;
    int z_accel;
    int temperature;
    int x_gyro;
    int y_gyro;
    int z_gyro;
  } value;
};

// Global constants
const unsigned long TIMEOUT = 18700;
//int rcpin1 = 7;
//int rcpin2 = 8;
int rcpin3 = 7;
//int rcpin4 = 10;
```

```
//int rcpin5 = 11;
//int rcpin6 = 12;
int maxthrot = 255;
int minthrot = 0;
int const nullthrot = 0;
int const chan3min = 1755;
int const chan3max = 909;

// Global function prototypes
void set_last_read_angle_data(unsigned long time, float x, float y, float z, float x_gyro, float
y_gyro, float z_gyro, float kx, float ky, float kz);
int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr);
void calibrate_sensors();

inline unsigned long get_last_time() {
  return last_read_time;
}
inline float get_last_x_angle() {
  return last_x_angle;
}
inline float get_last_y_angle() {
  return last_y_angle;
}
inline float get_last_z_angle() {
  return last_z_angle;
}
inline float get_last_gyro_x_angle() {
  return last_gyro_x_angle;
}
inline float get_last_gyro_y_angle() {
  return last_gyro_y_angle;
}
inline float get_last_gyro_z_angle() {
  return last_gyro_z_angle;
}
inline float get_last_x_kalangle() {
  return last_x_kalangle;
}
inline float get_last_y_kalangle() {
  return last_y_kalangle;
}
inline float get_last_z_kalangle() {
  return last_z_kalangle;
}

unsigned int cutoff(int rawin, int minthrot, int maxthrot);
unsigned int RCread(int pinin, int chanmin, int chanmax);

/********
* SETUP *
*********/

void setup()
{
  int error;
  uint8_t c;
  accel_t_gyro_union accel_t_gyro;
  Serial.begin(9600);
  Wire.begin();
  error = MPU6050_read (MPU6050_WHO_AM_I, &c, 1);
  error = MPU6050_read (MPU6050_PWR_MGMT_2, &c, 1);
  MPU6050_write_reg (MPU6050_PWR_MGMT_1, 0);
  calibrate_sensors();
  set_last_read_angle_data(millis(), 0, 0, 0, 0, 0, 0, 0, 0);
  float accel_x = accel_t_gyro.value.x_accel;
  float accel_y = accel_t_gyro.value.y_accel;
  float accel_z = accel_t_gyro.value.z_accel;
  float RADIANS_TO_DEGREES = 180 / 3.14159;
  float accel_angle_y = atan(-1 * accel_x / sqrt(pow(accel_y, 2) + pow(accel_z, 2))) *
RADIANS_TO_DEGREES;
  float accel_angle_x = atan(accel_y / sqrt(pow(accel_x, 2) + pow(accel_z, 2))) *
RADIANS_TO_DEGREES;
  float accel_angle_z = 0;
  kalmanX.setAngle(accel_angle_x); // Set starting angle
  kalmanY.setAngle(accel_angle_y);
  pinMode(ledPin, OUTPUT);
  for (temp = 0; temp < 4; temp++)
  {
    motor[temp] = 0;
  }
  min_speed = 0;
  max_speed = 0;

  PIDroll.ChangeParameters(ROLL_PID_KP, ROLL_PID_KI, ROLL_PID_KD, ROLL_PID_MIN, ROLL_PID_MAX);
  PIDpitch.ChangeParameters(PITCH_PID_KP, PITCH_PID_KI, PITCH_PID_KD, PITCH_PID_MIN,
PITCH_PID_MAX);
  PIDyaw.ChangeParameters(YAW_PID_KP, YAW_PID_KI, YAW_PID_KD, YAW_PID_MIN, YAW_PID_MAX);
  PIDangleX.ChangeParameters(ANGLEX_KP, ANGLEX_KI, ANGLEX_KD, ANGLEX_MIN, ANGLEX_MAX);
  PIDangleY.ChangeParameters(ANGLEY_KP, ANGLEY_KI, ANGLEY_KD, ANGLEY_MIN, ANGLEY_MAX);

  PIDangleX.resetI();
  PIDangleY.resetI();
  pinMode(rcpin3, INPUT);  // 970 - 1330; U - D
}

/************
* Main loop *
*************/
void loop()
{
  int error;
  double dT;
  accel_t_gyro_union accel_t_gyro;

  float kalAngleX, kalAngleY, kalAngleZ; // Calculate the angle using a Kalman filter
  float kalAngleX_last, kalAngleY_last, kalAngleZ_last;

  // Read the raw values.
  error = read_gyro_accel_vals((uint8_t*) &accel_t_gyro);

  // Get the time of reading for rotation computations
  unsigned long t_now = millis();


  // Convert gyro values to degrees/sec
  float FS_SEL = 131;

  float gyro_x = (accel_t_gyro.value.x_gyro - base_x_gyro) / FS_SEL;
  float gyro_y = (accel_t_gyro.value.y_gyro - base_y_gyro) / FS_SEL;
  float gyro_z = (accel_t_gyro.value.z_gyro - base_z_gyro) / FS_SEL;


  // Get raw acceleration values
  //float G_CONVERT = 16384;
  float accel_x = accel_t_gyro.value.x_accel;
  float accel_y = accel_t_gyro.value.y_accel;
  float accel_z = accel_t_gyro.value.z_accel;

  // Get angle values from accelerometer
  float RADIANS_TO_DEGREES = 180 / 3.14159;
  //  float accel_vector_length = sqrt(pow(accel_x,2) + pow(accel_y,2) + pow(accel_z,2));
  float accel_angle_y = atan(-1 * accel_x / sqrt(pow(accel_y, 2) + pow(accel_z, 2))) *
RADIANS_TO_DEGREES;
  float accel_angle_x = atan(accel_y / sqrt(pow(accel_x, 2) + pow(accel_z, 2))) *
RADIANS_TO_DEGREES;

  float accel_angle_z = 0;

  // Compute the (filtered) gyro angles
  float dt = (t_now - get_last_time()) / 1000.0;
  float gyro_angle_x = gyro_x * dt + get_last_x_angle();
  float gyro_angle_y = gyro_y * dt + get_last_y_angle();
  float gyro_angle_z = gyro_z * dt + get_last_z_angle();

  // Compute the drifting gyro angles
  float unfiltered_gyro_angle_x = gyro_x * dt + get_last_gyro_x_angle();
  float unfiltered_gyro_angle_y = gyro_y * dt + get_last_gyro_y_angle();
  float unfiltered_gyro_angle_z = gyro_z * dt + get_last_gyro_z_angle();

  // Apply the complementary filter to figure out the change in angle - choice of alpha is
  // estimated now.  Alpha depends on the sampling rate...
  float alpha = 0.94; //change was 0.96
  float angle_x = alpha * gyro_angle_x + (1.0 - alpha) * accel_angle_x;
  float angle_y = alpha * gyro_angle_y + (1.0 - alpha) * accel_angle_y;
  float angle_z = gyro_angle_z;  //Accelerometer doesn't give z-angle


  //KALMAN filter
  kalAngleX = kalmanX.getAngle(accel_angle_x, gyro_x, dt);
  kalAngleY = kalmanY.getAngle(accel_angle_y, gyro_y, dt);
  kalAngleZ = gyro_angle_z;

  if (millis() > .5)
  {
    Serial.print("$x ");
    Serial.println(kalAngleX);
    Serial.print("$y ");
    Serial.println(kalAngleY);
    Serial.print("$z ");
    Serial.print(kalAngleZ);
    Serial.println();
    Serial.print("$q ");
    Serial.println(millis());
  }

  // Update the saved data with the latest values
  set_last_read_angle_data(t_now, angle_x, angle_y, angle_z, unfiltered_gyro_angle_x,
unfiltered_gyro_angle_y, unfiltered_gyro_angle_z, kalAngleX, kalAngleY, kalAngleZ);

  // Outdated serial commands...
  if (Serial.available() > 0)
  {
    incomingByte = Serial.read();
    if (incomingByte == 'H')
    {
      digitalWrite(ledPin, HIGH);
    }
    if (incomingByte == 'L')
    {
      digitalWrite(ledPin, LOW);
    }
    if (incomingByte == 'A')
    {
      //      throttle = 100;
      min_speed = 100;
      max_speed = 100;
    }
    if (incomingByte == 'O')
    {
      //      throttle = 220;
      min_speed = 150;
      max_speed = 254;
    }
    if (incomingByte == 'F')
    {
      //      throttle = 0;
      min_speed = 0;
      max_speed = 0;
    }

  }

  //prevent motors from powering up for one minute from start up...
  //this gives the gyro some time to start up
  if (millis() < 10000)
  {
    Serial.print(millis());
    Serial.print('\n');
    digitalWrite(ledPin, LOW);
  }
  else
  {
    digitalWrite(ledPin, HIGH);
    if (event_armed == 0)
    {
      //      Serial.println("arming0: "); Serial.print(RCread(rcpin3, chan3min, chan3max));
      if (RCread(rcpin3, chan3min, chan3max) < 5 || event1 == 1)
      {
        //        Serial.println("arming1");
        event1 = 1;
        if (RCread(rcpin3, chan3min, chan3max) > 250 || event2 == 1)
        {
          //          Serial.println("arming2");
          event2 = 1;
          if (RCread(rcpin3, chan3min, chan3max) < 5) event3 = 1;
        }
      }
      if (event3 == 1)
      {
        Serial.println("armed");
        max_speed = 100;
        min_speed = 100;
        maxthrot = 100;
        minthrot = 100;
        event_armed = 1;
        analogWrite(8 , 100);
        analogWrite(9 , 100);
        analogWrite(10, 100);
        analogWrite(11, 100);
        delay(2000);
      }
    }
    else
    {
      max_speed = 254;
      min_speed = 150;
      maxthrot = 254;
      minthrot = 150;

    }

    int PIDroll_val = (int)PIDroll.Compute((float)kalAngleY);
    int PIDpitch_val = (int)PIDpitch.Compute((float)kalAngleX);
    int PIDyaw_val = (int)PIDyaw.Compute((float)kalAngleZ);
    //    Serial.print("PIDroll_val: ");
    //    Serial.print(PIDroll_val);
    //    Serial.print('\n');
    //    Serial.print("PIDpitch_val: ");
    //    Serial.print(PIDpitch_val);
    //    Serial.print('\n');
    //    Serial.print("PIDyaw_val: ");
    //    Serial.print(kalAngleZ);
    //    Serial.print('\n');
    motor[1] = RCread(rcpin3, chan3min, chan3max) + PIDroll_val - PIDpitch_val + PIDyaw_val; //11
    motor[2] = RCread(rcpin3, chan3min, chan3max) - PIDroll_val + PIDpitch_val - PIDyaw_val; //9
    motor[0] = RCread(rcpin3, chan3min, chan3max) + PIDroll_val - PIDpitch_val + PIDyaw_val; // 8
    motor[3] = RCread(rcpin3, chan3min, chan3max) - PIDroll_val + PIDpitch_val + PIDyaw_val; //10

    //prevents motors from firing at incorrect times or powering off in flight
    for (temp = 0; temp < 4; temp++)
    {
      if (motor[temp] > max_speed)
      {
        motor[temp] = max_speed;
      }
      else if (motor[temp] < min_speed)
      {
        motor[temp] = min_speed;
      }
```

```
    }

    //    Serial.print("Motor 11: ");
    //    Serial.print(int(round(motor[1])));
    //    Serial.print("|");
    //    Serial.print(int(event1));
    //    Serial.print("|");
    //    Serial.print(int(event2));
    //    Serial.print("|");
    //    Serial.print(int(event3));
    //    Serial.print("|");
    //    Serial.print(int(event_armed));
    //    Serial.print("Motor 9: ");
    //    Serial.print(int(round(motor[2])));
    //    Serial.print("|");
    //    Serial.print(int(event1));
    //    Serial.print("|");
    //    Serial.print(int(event2));
    //    Serial.print("|");
    //    Serial.print(int(event3));
    //    Serial.print("|");
    //    Serial.print(int(event_armed));
    //
    //    Serial.print('\n');


    analogWrite(8 , int(round(motor[0])));
    analogWrite(9 , int(round(motor[2])));
    analogWrite(10, int(round(motor[3])));
    analogWrite(11, int(round(motor[1])));
  }
}


// --------------------------------------------------------
// MPU6050_read
//
// This is a common function to read multiple bytes
// from an I2C device.
//
// It uses the boolean parameter for Wire.endTransMission()
// to be able to hold or release the I2C-bus.
// This is implemented in Arduino 1.0.1.
//
// Only this function is used to read.
// There is no function for a single byte.
//
int MPU6050_read(int start, uint8_t *buffer, int size)
{
  int i, n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);
  if (n != 1)
    return (-10);

  n = Wire.endTransmission(false);    // hold the I2C-bus
  if (n != 0)
```

```
    return (n);

  // Third parameter is true: relase I2C-bus after data is read.
  Wire.requestFrom(MPU6050_I2C_ADDRESS, size, true);
  i = 0;
  while (Wire.available() && i < size)
  {
    buffer[i++] = Wire.read();
  }
  if ( i != size)
    return (-11);

  return (0);  // return : no error
}


// --------------------------------------------------------
// MPU6050_write
//
// This is a common function to write multiple bytes to an I2C device.
//
// If only a single register is written,
// use the function MPU_6050_write_reg().
//
// Parameters:
//   start : Start address, use a define for the register
//   pData : A pointer to the data to write.
//   size  : The number of bytes to write.
//
// If only a single register is written, a pointer
// to the data has to be used, and the size is
// a single byte:
//   int data = 0;        // the data to write
//   MPU6050_write (MPU6050_PWR_MGMT_1, &c, 1);
//
int MPU6050_write(int start, const uint8_t *pData, int size)
{
  int n, error;

  Wire.beginTransmission(MPU6050_I2C_ADDRESS);
  n = Wire.write(start);        // write the start address
  if (n != 1)
    return (-20);

  n = Wire.write(pData, size);  // write data bytes
  if (n != size)
    return (-21);

  error = Wire.endTransmission(true); // release the I2C-bus
  if (error != 0)
    return (error);

  return (0);         // return : no error
}

// --------------------------------------------------------
// MPU6050_write_reg
//
```

```
// An extra function to write a single register.
// It is just a wrapper around the MPU_6050_write()
// function, and it is only a convenient function
// to make it easier to write a single register.
//
int MPU6050_write_reg(int reg, uint8_t data)
{
  int error;

  error = MPU6050_write(reg, &data, 1);

  return (error);
}

void set_last_read_angle_data(unsigned long time, float x, float y, float z, float x_gyro, float
y_gyro, float z_gyro, float kx, float ky, float kz) {
  last_read_time = time;
  last_x_angle = x;
  last_y_angle = y;
  last_z_angle = z;
  last_gyro_x_angle = x_gyro;
  last_gyro_y_angle = y_gyro;
  last_gyro_z_angle = z_gyro;
  last_x_kalangle = kx;
  last_y_kalangle = ky;
  last_z_kalangle = kz;
}

int read_gyro_accel_vals(uint8_t* accel_t_gyro_ptr) {
  // Read the raw values.
  // Read 14 bytes at once,
  // containing acceleration, temperature and gyro.
  // With the default settings of the MPU-6050,
  // there is no filter enabled, and the values
  // are not very stable.  Returns the error value

  accel_t_gyro_union* accel_t_gyro = (accel_t_gyro_union *) accel_t_gyro_ptr;

  int error = MPU6050_read (MPU6050_ACCEL_XOUT_H, (uint8_t *) accel_t_gyro,
sizeof(*accel_t_gyro));

  // Swap all high and low bytes.
  // After this, the registers values are swapped,
  // so the structure name like x_accel_l does no
  // longer contain the lower byte.
  uint8_t swap;
#define SWAP(x,y) swap = x; x = y; y = swap

  SWAP ((*accel_t_gyro).reg.x_accel_h, (*accel_t_gyro).reg.x_accel_l);
  SWAP ((*accel_t_gyro).reg.y_accel_h, (*accel_t_gyro).reg.y_accel_l);
  SWAP ((*accel_t_gyro).reg.z_accel_h, (*accel_t_gyro).reg.z_accel_l);
  SWAP ((*accel_t_gyro).reg.t_h, (*accel_t_gyro).reg.t_l);
  SWAP ((*accel_t_gyro).reg.x_gyro_h, (*accel_t_gyro).reg.x_gyro_l);
  SWAP ((*accel_t_gyro).reg.y_gyro_h, (*accel_t_gyro).reg.y_gyro_l);
  SWAP ((*accel_t_gyro).reg.z_gyro_h, (*accel_t_gyro).reg.z_gyro_l);

  return error;
}
```

```
// The sensor should be motionless on a horizontal surface
//  while calibration is happening
void calibrate_sensors() {
  int                   num_readings = 10;
  float                 x_accel = 0;
  float                 y_accel = 0;
  float                 z_accel = 0;
  float                 x_gyro = 0;
  float                 y_gyro = 0;
  float                 z_gyro = 0;
  accel_t_gyro_union    accel_t_gyro;

  // Calibration start
  // Discard the first set of values read from the IMU
  read_gyro_accel_vals((uint8_t *) &accel_t_gyro);

  // Read and average the raw values from the IMU
  for (int i = 0; i < num_readings; i++) {
    read_gyro_accel_vals((uint8_t *) &accel_t_gyro);
    x_accel += accel_t_gyro.value.x_accel;
    y_accel += accel_t_gyro.value.y_accel;
    z_accel += accel_t_gyro.value.z_accel;
    x_gyro += accel_t_gyro.value.x_gyro;
    y_gyro += accel_t_gyro.value.y_gyro;
    z_gyro += accel_t_gyro.value.z_gyro;
    delay(100);
  }
  x_accel /= num_readings;
  y_accel /= num_readings;
  z_accel /= num_readings;
  x_gyro /= num_readings;
  y_gyro /= num_readings;
  z_gyro /= num_readings;

  // Store the raw calibration values globally
  base_x_accel = x_accel;
  base_y_accel = y_accel;
  base_z_accel = z_accel;
  base_x_gyro = x_gyro;
  base_y_gyro = y_gyro;
  base_z_gyro = z_gyro;

  // Calibration finished
}

unsigned int cutoff(int rawin, int minthrot, int maxthrot)
{
  if (rawin < minthrot)
  {
    rawin = minthrot;
  }
  else if (rawin > maxthrot)
  {
    rawin = maxthrot;
  }
  return rawin;
}
```

```
unsigned int RCread(int pinin, int chanmin, int chanmax)
{
  /*

  */
  unsigned int rawin = pulseIn(pinin, HIGH, TIMEOUT);
  if (rawin != 0)
  {
    //    Serial.print(rawin);

    rawin = map(rawin, chanmin, chanmax, minthrot, maxthrot);
    rawin = cutoff(rawin, minthrot, maxthrot);
    //    Serial.println(rawin);
    //    Serial.print(", ");
  }
  else
  {
    //    Serial.print("0:0, ");
    rawin = nullthrot;
  }
  return rawin;
}


//MPU_6050.h
// MPU-6050 Accelerometer + Gyro + Arduino Uno
// -----------------------------
//
//
// June 2012
// Open Source / Public Domain
//
// Using Arduino 1.0.1
// It will not work with an older version,
// since Wire.endTransmission() uses a parameter
// to hold or release the I2C bus.
//
// Documentation:
// - The InvenSense documents:
//   - "MPU-6000 and MPU-6050 Product Specification",
//     PS-MPU-6000A.pdf
//   - "MPU-6000 and MPU-6050 Register Map and Descriptions",
//     RM-MPU-6000A.pdf or RS-MPU-6000A.pdf
//   - "MPU-6000/MPU-6050 9-Axis Evaluation Board User Guide"
//     AN-MPU-6000EVB.pdf
//
// The accuracy is 16-bits.
//
// Temperature sensor from -40 to +85 degrees Celsius
//   340 per degrees, -512 at 35 degrees.
//
// At power-up, all registers are zero, except these two:
//      Register 0x6B (PWR_MGMT_2) = 0x40  (I read zero).
//      Register 0x75 (WHO_AM_I)   = 0x68.
//

// Register names according to the datasheet.
// According to the InvenSense document
// "MPU-6000 and MPU-6050 Register Map
// and Descriptions Revision 3.2", there are no registers
// at 0x02 ... 0x18, but according other information
// the registers in that unknown area are for gain
// and offsets.
//
#define MPU6050_AUX_VDDIO          0x01   // R/W
#define MPU6050_SMPLRT_DIV         0x19   // R/W
#define MPU6050_CONFIG             0x1A   // R/W
#define MPU6050_GYRO_CONFIG        0x1B   // R/W
#define MPU6050_ACCEL_CONFIG       0x1C   // R/W
#define MPU6050_FF_THR             0x1D   // R/W
#define MPU6050_FF_DUR             0x1E   // R/W
#define MPU6050_MOT_THR            0x1F   // R/W
#define MPU6050_MOT_DUR            0x20   // R/W
#define MPU6050_ZRMOT_THR          0x21   // R/W
#define MPU6050_ZRMOT_DUR          0x22   // R/W
#define MPU6050_FIFO_EN            0x23   // R/W
#define MPU6050_I2C_MST_CTRL       0x24   // R/W
#define MPU6050_I2C_SLV0_ADDR      0x25   // R/W
#define MPU6050_I2C_SLV0_REG       0x26   // R/W
#define MPU6050_I2C_SLV0_CTRL      0x27   // R/W
#define MPU6050_I2C_SLV1_ADDR      0x28   // R/W
#define MPU6050_I2C_SLV1_REG       0x29   // R/W
#define MPU6050_I2C_SLV1_CTRL      0x2A   // R/W
#define MPU6050_I2C_SLV2_ADDR      0x2B   // R/W
#define MPU6050_I2C_SLV2_REG       0x2C   // R/W
#define MPU6050_I2C_SLV2_CTRL      0x2D   // R/W
#define MPU6050_I2C_SLV3_ADDR      0x2E   // R/W
#define MPU6050_I2C_SLV3_REG       0x2F   // R/W
#define MPU6050_I2C_SLV3_CTRL      0x30   // R/W
#define MPU6050_I2C_SLV4_ADDR      0x31   // R/W
#define MPU6050_I2C_SLV4_REG       0x32   // R/W
#define MPU6050_I2C_SLV4_DO        0x33   // R/W
#define MPU6050_I2C_SLV4_CTRL      0x34   // R/W
#define MPU6050_I2C_SLV4_DI        0x35   // R
#define MPU6050_I2C_MST_STATUS     0x36   // R
#define MPU6050_INT_PIN_CFG        0x37   // R/W
#define MPU6050_INT_ENABLE         0x38   // R/W
#define MPU6050_INT_STATUS         0x3A   // R
#define MPU6050_ACCEL_XOUT_H       0x3B   // R
#define MPU6050_ACCEL_XOUT_L       0x3C   // R
#define MPU6050_ACCEL_YOUT_H       0x3D   // R
#define MPU6050_ACCEL_YOUT_L       0x3E   // R
#define MPU6050_ACCEL_ZOUT_H       0x3F   // R
#define MPU6050_ACCEL_ZOUT_L       0x40   // R
#define MPU6050_TEMP_OUT_H         0x41   // R
#define MPU6050_TEMP_OUT_L         0x42   // R
#define MPU6050_GYRO_XOUT_H        0x43   // R
#define MPU6050_GYRO_XOUT_L        0x44   // R
#define MPU6050_GYRO_YOUT_H        0x45   // R
#define MPU6050_GYRO_YOUT_L        0x46   // R
#define MPU6050_GYRO_ZOUT_H        0x47   // R
#define MPU6050_GYRO_ZOUT_L        0x48   // R
#define MPU6050_EXT_SENS_DATA_00   0x49   // R
#define MPU6050_EXT_SENS_DATA_01   0x4A   // R
#define MPU6050_EXT_SENS_DATA_02   0x4B   // R
```

```
#define MPU6050_EXT_SENS_DATA_03   0x4C   // R
#define MPU6050_EXT_SENS_DATA_04   0x4D   // R
#define MPU6050_EXT_SENS_DATA_05   0x4E   // R
#define MPU6050_EXT_SENS_DATA_06   0x4F   // R
#define MPU6050_EXT_SENS_DATA_07   0x50   // R
#define MPU6050_EXT_SENS_DATA_08   0x51   // R
#define MPU6050_EXT_SENS_DATA_09   0x52   // R
#define MPU6050_EXT_SENS_DATA_10   0x53   // R
#define MPU6050_EXT_SENS_DATA_11   0x54   // R
#define MPU6050_EXT_SENS_DATA_12   0x55   // R
#define MPU6050_EXT_SENS_DATA_13   0x56   // R
#define MPU6050_EXT_SENS_DATA_14   0x57   // R
#define MPU6050_EXT_SENS_DATA_15   0x58   // R
#define MPU6050_EXT_SENS_DATA_16   0x59   // R
#define MPU6050_EXT_SENS_DATA_17   0x5A   // R
#define MPU6050_EXT_SENS_DATA_18   0x5B   // R
#define MPU6050_EXT_SENS_DATA_19   0x5C   // R
#define MPU6050_EXT_SENS_DATA_20   0x5D   // R
#define MPU6050_EXT_SENS_DATA_21   0x5E   // R
#define MPU6050_EXT_SENS_DATA_22   0x5F   // R
#define MPU6050_EXT_SENS_DATA_23   0x60   // R
#define MPU6050_MOT_DETECT_STATUS  0x61   // R
#define MPU6050_I2C_SLV0_DO        0x63   // R/W
#define MPU6050_I2C_SLV1_DO        0x64   // R/W
#define MPU6050_I2C_SLV2_DO        0x65   // R/W
#define MPU6050_I2C_SLV3_DO        0x66   // R/W
#define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
#define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
#define MPU6050_MOT_DETECT_CTRL    0x69   // R/W
#define MPU6050_USER_CTRL          0x6A   // R/W
#define MPU6050_PWR_MGMT_1         0x6B   // R/W
#define MPU6050_PWR_MGMT_2         0x6C   // R/W
#define MPU6050_FIFO_COUNTH        0x72   // R/W
#define MPU6050_FIFO_COUNTL        0x73   // R/W
#define MPU6050_FIFO_R_W           0x74   // R/W
#define MPU6050_WHO_AM_I           0x75   // R

// Defines for the bits, to be able to change
// between bit number and binary definition.
// By using the bit number, programming the sensor
// is like programming the AVR microcontroller.
// But instead of using "(1<<X)", or "_BV(X)",
// the Arduino "bit(X)" is used.
#define MPU6050_D0 0
#define MPU6050_D1 1
#define MPU6050_D2 2
#define MPU6050_D3 3
#define MPU6050_D4 4
#define MPU6050_D5 5
#define MPU6050_D6 6
#define MPU6050_D7 7

// AUX_VDDIO Register
#define MPU6050_AUX_VDDIO MPU6050_D7  // I2C high: 1=VDD, 0=VLOGIC

// CONFIG Register
// DLPF is Digital Low Pass Filter for both gyro and accelerometers.
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_DLPF_CFG0     MPU6050_D0
#define MPU6050_DLPF_CFG1     MPU6050_D1
#define MPU6050_DLPF_CFG2     MPU6050_D2
#define MPU6050_EXT_SYNC_SET0 MPU6050_D3
#define MPU6050_EXT_SYNC_SET1 MPU6050_D4
#define MPU6050_EXT_SYNC_SET2 MPU6050_D5

// Combined definitions for the EXT_SYNC_SET values
#define MPU6050_EXT_SYNC_SET_0 (0)
#define MPU6050_EXT_SYNC_SET_1 (bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_2 (bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_3 (bit(MPU6050_EXT_SYNC_SET1)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_4 (bit(MPU6050_EXT_SYNC_SET2))
#define MPU6050_EXT_SYNC_SET_5 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET0))
#define MPU6050_EXT_SYNC_SET_6 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1))
#define MPU6050_EXT_SYNC_SET_7 (bit(MPU6050_EXT_SYNC_SET2)|bit(MPU6050_EXT_SYNC_SET1)|
bit(MPU6050_EXT_SYNC_SET0))

// Alternative names for the combined definitions.
#define MPU6050_EXT_SYNC_DISABLED     MPU6050_EXT_SYNC_SET_0
#define MPU6050_EXT_SYNC_TEMP_OUT_L   MPU6050_EXT_SYNC_SET_1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L  MPU6050_EXT_SYNC_SET_2
#define MPU6050_EXT_SYNC_GYRO_YOUT_L  MPU6050_EXT_SYNC_SET_3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L  MPU6050_EXT_SYNC_SET_4
#define MPU6050_EXT_SYNC_ACCEL_XOUT_L MPU6050_EXT_SYNC_SET_5
#define MPU6050_EXT_SYNC_ACCEL_YOUT_L MPU6050_EXT_SYNC_SET_6
#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L MPU6050_EXT_SYNC_SET_7

// Combined definitions for the DLPF_CFG values
#define MPU6050_DLPF_CFG_0 (0)
#define MPU6050_DLPF_CFG_1 (bit(MPU6050_DLPF_CFG0))
#define MPU6050_DLPF_CFG_2 (bit(MPU6050_DLPF_CFG1))
#define MPU6050_DLPF_CFG_3 (bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))
#define MPU6050_DLPF_CFG_4 (bit(MPU6050_DLPF_CFG2))
#define MPU6050_DLPF_CFG_5 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG0))
#define MPU6050_DLPF_CFG_6 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1))
#define MPU6050_DLPF_CFG_7 (bit(MPU6050_DLPF_CFG2)|bit(MPU6050_DLPF_CFG1)|bit(MPU6050_DLPF_CFG0))

// Alternative names for the combined definitions
// This name uses the bandwidth (Hz) for the accelometer,
// for the gyro the bandwidth is almost the same.
#define MPU6050_DLPF_260HZ    MPU6050_DLPF_CFG_0
#define MPU6050_DLPF_184HZ    MPU6050_DLPF_CFG_1
#define MPU6050_DLPF_94HZ     MPU6050_DLPF_CFG_2
#define MPU6050_DLPF_44HZ     MPU6050_DLPF_CFG_3
#define MPU6050_DLPF_21HZ     MPU6050_DLPF_CFG_4
#define MPU6050_DLPF_10HZ     MPU6050_DLPF_CFG_5
#define MPU6050_DLPF_5HZ      MPU6050_DLPF_CFG_6
#define MPU6050_DLPF_RESERVED MPU6050_DLPF_CFG_7

// GYRO_CONFIG Register
// The XG_ST, YG_ST, ZG_ST are bits for selftest.
// The FS_SEL sets the range for the gyro.
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_FS_SEL0 MPU6050_D3
#define MPU6050_FS_SEL1 MPU6050_D4
```

```
#define MPU6050_ZG_ST   MPU6050_D5
#define MPU6050_YG_ST   MPU6050_D6
#define MPU6050_XG_ST   MPU6050_D7

// Combined definitions for the FS_SEL values
#define MPU6050_FS_SEL_0 (0)
#define MPU6050_FS_SEL_1 (bit(MPU6050_FS_SEL0))
#define MPU6050_FS_SEL_2 (bit(MPU6050_FS_SEL1))
#define MPU6050_FS_SEL_3 (bit(MPU6050_FS_SEL1)|bit(MPU6050_FS_SEL0))

// Alternative names for the combined definitions
// The name uses the range in degrees per second.
#define MPU6050_FS_SEL_250  MPU6050_FS_SEL_0
#define MPU6050_FS_SEL_500  MPU6050_FS_SEL_1
#define MPU6050_FS_SEL_1000 MPU6050_FS_SEL_2
#define MPU6050_FS_SEL_2000 MPU6050_FS_SEL_3

// ACCEL_CONFIG Register
// The XA_ST, YA_ST, ZA_ST are bits for selftest.
// The AFS_SEL sets the range for the accelerometer.
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_ACCEL_HPF0 MPU6050_D0
#define MPU6050_ACCEL_HPF1 MPU6050_D1
#define MPU6050_ACCEL_HPF2 MPU6050_D2
#define MPU6050_AFS_SEL0   MPU6050_D3
#define MPU6050_AFS_SEL1   MPU6050_D4
#define MPU6050_ZA_ST      MPU6050_D5
#define MPU6050_YA_ST      MPU6050_D6
#define MPU6050_XA_ST      MPU6050_D7

// Combined definitions for the ACCEL_HPF values
#define MPU6050_ACCEL_HPF_0 (0)
#define MPU6050_ACCEL_HPF_1 (bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_2 (bit(MPU6050_ACCEL_HPF1))
#define MPU6050_ACCEL_HPF_3 (bit(MPU6050_ACCEL_HPF1)|bit(MPU6050_ACCEL_HPF0))
#define MPU6050_ACCEL_HPF_4 (bit(MPU6050_ACCEL_HPF2))
#define MPU6050_ACCEL_HPF_7 (bit(MPU6050_ACCEL_HPF2)|bit(MPU6050_ACCEL_HPF1)|
bit(MPU6050_ACCEL_HPF0))

// Alternative names for the combined definitions
// The name uses the Cut-off frequency.
#define MPU6050_ACCEL_HPF_RESET  MPU6050_ACCEL_HPF_0
#define MPU6050_ACCEL_HPF_5HZ    MPU6050_ACCEL_HPF_1
#define MPU6050_ACCEL_HPF_2_5HZ  MPU6050_ACCEL_HPF_2
#define MPU6050_ACCEL_HPF_1_25HZ MPU6050_ACCEL_HPF_3
#define MPU6050_ACCEL_HPF_0_63HZ MPU6050_ACCEL_HPF_4
#define MPU6050_ACCEL_HPF_HOLD   MPU6050_ACCEL_HPF_7

// Combined definitions for the AFS_SEL values
#define MPU6050_AFS_SEL_0 (0)
#define MPU6050_AFS_SEL_1 (bit(MPU6050_AFS_SEL0))
#define MPU6050_AFS_SEL_2 (bit(MPU6050_AFS_SEL1))
#define MPU6050_AFS_SEL_3 (bit(MPU6050_AFS_SEL1)|bit(MPU6050_AFS_SEL0))

// Alternative names for the combined definitions
// The name uses the full scale range for the accelerometer.
#define MPU6050_AFS_SEL_2G  MPU6050_AFS_SEL_0
```

```
#define MPU6050_AFS_SEL_4G  MPU6050_AFS_SEL_1
#define MPU6050_AFS_SEL_8G  MPU6050_AFS_SEL_2
#define MPU6050_AFS_SEL_16G MPU6050_AFS_SEL_3

// FIFO_EN Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_SLV0_FIFO_EN  MPU6050_D0
#define MPU6050_SLV1_FIFO_EN  MPU6050_D1
#define MPU6050_SLV2_FIFO_EN  MPU6050_D2
#define MPU6050_ACCEL_FIFO_EN MPU6050_D3
#define MPU6050_ZG_FIFO_EN    MPU6050_D4
#define MPU6050_YG_FIFO_EN    MPU6050_D5
#define MPU6050_XG_FIFO_EN    MPU6050_D6
#define MPU6050_TEMP_FIFO_EN  MPU6050_D7

// I2C_MST_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_MST_CLK0  MPU6050_D0
#define MPU6050_I2C_MST_CLK1  MPU6050_D1
#define MPU6050_I2C_MST_CLK2  MPU6050_D2
#define MPU6050_I2C_MST_CLK3  MPU6050_D3
#define MPU6050_I2C_MST_P_NSR MPU6050_D4
#define MPU6050_SLV_3_FIFO_EN MPU6050_D5
#define MPU6050_WAIT_FOR_ES   MPU6050_D6
#define MPU6050_MULT_MST_EN   MPU6050_D7

// Combined definitions for the I2C_MST_CLK
#define MPU6050_I2C_MST_CLK_0 (0)
#define MPU6050_I2C_MST_CLK_1  (bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_2  (bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_3  (bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_4  (bit(MPU6050_I2C_MST_CLK2))
#define MPU6050_I2C_MST_CLK_5  (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_6  (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_7  (bit(MPU6050_I2C_MST_CLK2)|bit(MPU6050_I2C_MST_CLK1)|
bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_8  (bit(MPU6050_I2C_MST_CLK3))
#define MPU6050_I2C_MST_CLK_9  (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_10 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_11 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK1)|
bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_12 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2))
#define MPU6050_I2C_MST_CLK_13 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|
bit(MPU6050_I2C_MST_CLK0))
#define MPU6050_I2C_MST_CLK_14 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|
bit(MPU6050_I2C_MST_CLK1))
#define MPU6050_I2C_MST_CLK_15 (bit(MPU6050_I2C_MST_CLK3)|bit(MPU6050_I2C_MST_CLK2)|
bit(MPU6050_I2C_MST_CLK1)|bit(MPU6050_I2C_MST_CLK0))

// Alternative names for the combined definitions
// The names uses I2C Master Clock Speed in kHz.
#define MPU6050_I2C_MST_CLK_348KHZ MPU6050_I2C_MST_CLK_0
#define MPU6050_I2C_MST_CLK_333KHZ MPU6050_I2C_MST_CLK_1
#define MPU6050_I2C_MST_CLK_320KHZ MPU6050_I2C_MST_CLK_2
#define MPU6050_I2C_MST_CLK_308KHZ MPU6050_I2C_MST_CLK_3
#define MPU6050_I2C_MST_CLK_296KHZ MPU6050_I2C_MST_CLK_4
```

```
#define MPU6050_I2C_MST_CLK_286KHZ MPU6050_I2C_MST_CLK_5
#define MPU6050_I2C_MST_CLK_276KHZ MPU6050_I2C_MST_CLK_6
#define MPU6050_I2C_MST_CLK_267KHZ MPU6050_I2C_MST_CLK_7
#define MPU6050_I2C_MST_CLK_258KHZ MPU6050_I2C_MST_CLK_8
#define MPU6050_I2C_MST_CLK_500KHZ MPU6050_I2C_MST_CLK_9
#define MPU6050_I2C_MST_CLK_471KHZ MPU6050_I2C_MST_CLK_10
#define MPU6050_I2C_MST_CLK_444KHZ MPU6050_I2C_MST_CLK_11
#define MPU6050_I2C_MST_CLK_421KHZ MPU6050_I2C_MST_CLK_12
#define MPU6050_I2C_MST_CLK_400KHZ MPU6050_I2C_MST_CLK_13
#define MPU6050_I2C_MST_CLK_381KHZ MPU6050_I2C_MST_CLK_14
#define MPU6050_I2C_MST_CLK_364KHZ MPU6050_I2C_MST_CLK_15

// I2C_SLV0_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_RW MPU6050_D7

// I2C_SLV0_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV0_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV0_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV0_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV0_GRP     MPU6050_D4
#define MPU6050_I2C_SLV0_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV0_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV0_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV0_LEN_MASK 0x0F

// I2C_SLV1_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV1_RW MPU6050_D7

// I2C_SLV1_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV1_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV1_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV1_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV1_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV1_GRP     MPU6050_D4
#define MPU6050_I2C_SLV1_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV1_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV1_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV1_LEN_MASK 0x0F

// I2C_SLV2_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV2_RW MPU6050_D7

// I2C_SLV2_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV2_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV2_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV2_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV2_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV2_GRP     MPU6050_D4
#define MPU6050_I2C_SLV2_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV2_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV2_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV2_LEN_MASK 0x0F

// I2C_SLV3_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV3_RW MPU6050_D7

// I2C_SLV3_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV3_LEN0    MPU6050_D0
#define MPU6050_I2C_SLV3_LEN1    MPU6050_D1
#define MPU6050_I2C_SLV3_LEN2    MPU6050_D2
#define MPU6050_I2C_SLV3_LEN3    MPU6050_D3
#define MPU6050_I2C_SLV3_GRP     MPU6050_D4
#define MPU6050_I2C_SLV3_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV3_BYTE_SW MPU6050_D6
#define MPU6050_I2C_SLV3_EN      MPU6050_D7

// A mask for the length
#define MPU6050_I2C_SLV3_LEN_MASK 0x0F

// I2C_SLV4_ADDR Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV4_RW MPU6050_D7

// I2C_SLV4_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_MST_DLY0     MPU6050_D0
#define MPU6050_I2C_MST_DLY1     MPU6050_D1
#define MPU6050_I2C_MST_DLY2     MPU6050_D2
#define MPU6050_I2C_MST_DLY3     MPU6050_D3
#define MPU6050_I2C_MST_DLY4     MPU6050_D4
#define MPU6050_I2C_SLV4_REG_DIS MPU6050_D5
#define MPU6050_I2C_SLV4_INT_EN  MPU6050_D6
#define MPU6050_I2C_SLV4_EN      MPU6050_D7

// A mask for the delay
#define MPU6050_I2C_MST_DLY_MASK 0x1F

// I2C_MST_STATUS Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_NACK MPU6050_D0
```

```
#define MPU6050_I2C_SLV1_NACK MPU6050_D1
#define MPU6050_I2C_SLV2_NACK MPU6050_D2
#define MPU6050_I2C_SLV3_NACK MPU6050_D3
#define MPU6050_I2C_SLV4_NACK MPU6050_D4
#define MPU6050_I2C_LOST_ARB  MPU6050_D5
#define MPU6050_I2C_SLV4_DONE MPU6050_D6
#define MPU6050_PASS_THROUGH  MPU6050_D7

// I2C_PIN_CFG Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_CLKOUT_EN       MPU6050_D0
#define MPU6050_I2C_BYPASS_EN   MPU6050_D1
#define MPU6050_FSYNC_INT_EN    MPU6050_D2
#define MPU6050_FSYNC_INT_LEVEL MPU6050_D3
#define MPU6050_INT_RD_CLEAR    MPU6050_D4
#define MPU6050_LATCH_INT_EN    MPU6050_D5
#define MPU6050_INT_OPEN        MPU6050_D6
#define MPU6050_INT_LEVEL       MPU6050_D7

// INT_ENABLE Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_DATA_RDY_EN    MPU6050_D0
#define MPU6050_I2C_MST_INT_EN MPU6050_D3
#define MPU6050_FIFO_OFLOW_EN  MPU6050_D4
#define MPU6050_ZMOT_EN        MPU6050_D5
#define MPU6050_MOT_EN         MPU6050_D6
#define MPU6050_FF_EN          MPU6050_D7

// INT_STATUS Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_DATA_RDY_INT   MPU6050_D0
#define MPU6050_I2C_MST_INT    MPU6050_D3
#define MPU6050_FIFO_OFLOW_INT MPU6050_D4
#define MPU6050_ZMOT_INT       MPU6050_D5
#define MPU6050_MOT_INT        MPU6050_D6
#define MPU6050_FF_INT         MPU6050_D7

// MOT_DETECT_STATUS Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_MOT_ZRMOT MPU6050_D0
#define MPU6050_MOT_ZPOS  MPU6050_D2
#define MPU6050_MOT_ZNEG  MPU6050_D3
#define MPU6050_MOT_YPOS  MPU6050_D4
#define MPU6050_MOT_YNEG  MPU6050_D5
#define MPU6050_MOT_XPOS  MPU6050_D6
#define MPU6050_MOT_XNEG  MPU6050_D7

// IC2_MST_DELAY_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_I2C_SLV0_DLY_EN MPU6050_D0
#define MPU6050_I2C_SLV1_DLY_EN MPU6050_D1
#define MPU6050_I2C_SLV2_DLY_EN MPU6050_D2
#define MPU6050_I2C_SLV3_DLY_EN MPU6050_D3
```

```
#define MPU6050_I2C_SLV4_DLY_EN MPU6050_D4
#define MPU6050_DELAY_ES_SHADOW MPU6050_D7

// SIGNAL_PATH_RESET Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_TEMP_RESET  MPU6050_D0
#define MPU6050_ACCEL_RESET MPU6050_D1
#define MPU6050_GYRO_RESET  MPU6050_D2

// MOT_DETECT_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_MOT_COUNT0      MPU6050_D0
#define MPU6050_MOT_COUNT1      MPU6050_D1
#define MPU6050_FF_COUNT0       MPU6050_D2
#define MPU6050_FF_COUNT1       MPU6050_D3
#define MPU6050_ACCEL_ON_DELAY0 MPU6050_D4
#define MPU6050_ACCEL_ON_DELAY1 MPU6050_D5

// Combined definitions for the MOT_COUNT
#define MPU6050_MOT_COUNT_0 (0)
#define MPU6050_MOT_COUNT_1 (bit(MPU6050_MOT_COUNT0))
#define MPU6050_MOT_COUNT_2 (bit(MPU6050_MOT_COUNT1))
#define MPU6050_MOT_COUNT_3 (bit(MPU6050_MOT_COUNT1)|bit(MPU6050_MOT_COUNT0))

// Alternative names for the combined definitions
#define MPU6050_MOT_COUNT_RESET MPU6050_MOT_COUNT_0

// Combined definitions for the FF_COUNT
#define MPU6050_FF_COUNT_0 (0)
#define MPU6050_FF_COUNT_1 (bit(MPU6050_FF_COUNT0))
#define MPU6050_FF_COUNT_2 (bit(MPU6050_FF_COUNT1))
#define MPU6050_FF_COUNT_3 (bit(MPU6050_FF_COUNT1)|bit(MPU6050_FF_COUNT0))

// Alternative names for the combined definitions
#define MPU6050_FF_COUNT_RESET MPU6050_FF_COUNT_0

// Combined definitions for the ACCEL_ON_DELAY
#define MPU6050_ACCEL_ON_DELAY_0 (0)
#define MPU6050_ACCEL_ON_DELAY_1 (bit(MPU6050_ACCEL_ON_DELAY0))
#define MPU6050_ACCEL_ON_DELAY_2 (bit(MPU6050_ACCEL_ON_DELAY1))
#define MPU6050_ACCEL_ON_DELAY_3 (bit(MPU6050_ACCEL_ON_DELAY1)|bit(MPU6050_ACCEL_ON_DELAY0))

// Alternative names for the ACCEL_ON_DELAY
#define MPU6050_ACCEL_ON_DELAY_0MS MPU6050_ACCEL_ON_DELAY_0
#define MPU6050_ACCEL_ON_DELAY_1MS MPU6050_ACCEL_ON_DELAY_1
#define MPU6050_ACCEL_ON_DELAY_2MS MPU6050_ACCEL_ON_DELAY_2
#define MPU6050_ACCEL_ON_DELAY_3MS MPU6050_ACCEL_ON_DELAY_3

// USER_CTRL Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_SIG_COND_RESET MPU6050_D0
#define MPU6050_I2C_MST_RESET  MPU6050_D1
#define MPU6050_FIFO_RESET     MPU6050_D2
#define MPU6050_I2C_IF_DIS     MPU6050_D4   // must be 0 for MPU-6050
#define MPU6050_I2C_MST_EN     MPU6050_D5
```

```
#define MPU6050_FIFO_EN        MPU6050_D6

// PWR_MGMT_1 Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_CLKSEL0      MPU6050_D0
#define MPU6050_CLKSEL1      MPU6050_D1
#define MPU6050_CLKSEL2      MPU6050_D2
#define MPU6050_TEMP_DIS     MPU6050_D3    // 1: disable temperature sensor
#define MPU6050_CYCLE        MPU6050_D5    // 1: sample and sleep
#define MPU6050_SLEEP        MPU6050_D6    // 1: sleep mode
#define MPU6050_DEVICE_RESET MPU6050_D7    // 1: reset to default values

// Combined definitions for the CLKSEL
#define MPU6050_CLKSEL_0 (0)
#define MPU6050_CLKSEL_1 (bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_2 (bit(MPU6050_CLKSEL1))
#define MPU6050_CLKSEL_3 (bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_4 (bit(MPU6050_CLKSEL2))
#define MPU6050_CLKSEL_5 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL0))
#define MPU6050_CLKSEL_6 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1))
#define MPU6050_CLKSEL_7 (bit(MPU6050_CLKSEL2)|bit(MPU6050_CLKSEL1)|bit(MPU6050_CLKSEL0))

// Alternative names for the combined definitions
#define MPU6050_CLKSEL_INTERNAL    MPU6050_CLKSEL_0
#define MPU6050_CLKSEL_X           MPU6050_CLKSEL_1
#define MPU6050_CLKSEL_Y           MPU6050_CLKSEL_2
#define MPU6050_CLKSEL_Z           MPU6050_CLKSEL_3
#define MPU6050_CLKSEL_EXT_32KHZ   MPU6050_CLKSEL_4
#define MPU6050_CLKSEL_EXT_19_2MHZ MPU6050_CLKSEL_5
#define MPU6050_CLKSEL_RESERVED    MPU6050_CLKSEL_6
#define MPU6050_CLKSEL_STOP        MPU6050_CLKSEL_7

// PWR_MGMT_2 Register
// These are the names for the bits.
// Use these only with the bit() macro.
#define MPU6050_STBY_ZG       MPU6050_D0
#define MPU6050_STBY_YG       MPU6050_D1
#define MPU6050_STBY_XG       MPU6050_D2
#define MPU6050_STBY_ZA       MPU6050_D3
#define MPU6050_STBY_YA       MPU6050_D4
#define MPU6050_STBY_XA       MPU6050_D5
#define MPU6050_LP_WAKE_CTRL0 MPU6050_D6
#define MPU6050_LP_WAKE_CTRL1 MPU6050_D7

// Combined definitions for the LP_WAKE_CTRL
#define MPU6050_LP_WAKE_CTRL_0 (0)
#define MPU6050_LP_WAKE_CTRL_1 (bit(MPU6050_LP_WAKE_CTRL0))
#define MPU6050_LP_WAKE_CTRL_2 (bit(MPU6050_LP_WAKE_CTRL1))
#define MPU6050_LP_WAKE_CTRL_3 (bit(MPU6050_LP_WAKE_CTRL1)|bit(MPU6050_LP_WAKE_CTRL0))

// Alternative names for the combined definitions
// The names uses the Wake-up Frequency.
#define MPU6050_LP_WAKE_1_25HZ MPU6050_LP_WAKE_CTRL_0
#define MPU6050_LP_WAKE_2_5HZ  MPU6050_LP_WAKE_CTRL_1
#define MPU6050_LP_WAKE_5HZ    MPU6050_LP_WAKE_CTRL_2
#define MPU6050_LP_WAKE_10HZ   MPU6050_LP_WAKE_CTRL_3
```

```
// Default I2C address for the MPU-6050 is 0x68.
// But only if the AD0 pin is low.
// Some sensor boards have AD0 high, and the
// I2C address thus becomes 0x69.
#define MPU6050_I2C_ADDRESS 0x68



//////////////quadcopter_serial_sensor_visualizer.pde
// Modified from Graphing sketch
// Original code was Created 20 Apr 2005
// Updated 18 Jan 2008
// by Tom Igoe
// The example code that was modified is in the public domain.

import processing.serial.*;

boolean graphitnao;
Table table;
String tablelocation="data/data.csv";
int graph = 0;
PrintWriter csvoutput;

float[] data;

Serial myPort;        // The serial port
int xPos = 1;         // horizontal position of the graph
int numgraph = 3;      // how many graphs?
String serialconnection = "/dev/tty.usb"; //What your serial connection is. For usb:
"tty.usbmodem"
float timer = 0;
/*
//Smoothing test
 // Define the number of samples to keep track of.  The higher the number,
 // the more the readings will be smoothed, but the slower the output will
 // respond to the input.  Using a constant rather than a normal variable lets
 // use this value to determine the size of the readings array.
 int numReadings = 10;

 int[][] readings = new int[numgraph][numReadings];      // the readings from the analog input
 int index = 0;                  // the index of the current reading
 int total = 0;                  // the running total
 int average = 0;                // the average
 */


void setup () {
  // set the window size:
  size(1200, 900);
  // List all the available serial ports
  int l = Serial.list().length;
  String[] supercereal = new String[l];
  supercereal=Serial.list();

  data =  new float[numgraph+2];

  //This part finds the USB.
  int i=0;
  for (; i<l; i++)
  {
    String[] m2 = match(supercereal[i], serialconnection);
    println(supercereal[i]);
    if (m2 != null)
    {
      println("Found " + supercereal[i]);
      break;
    } else
      println("Nope!");
  }

  data[4]=0;
  myPort = new Serial(this, Serial.list()[i], 9600);
  // don't generate a serialEvent() unless you get a newline character:
  myPort.bufferUntil('\n');
  // set inital background:
  background(0);
  /*
  //Smoothing test
   // initialize all the readings to 0:
   for (int thisReading = 0; thisReading < numReadings; thisReading++)
   for (int thisgraph = 0; thisgraph < numgraph; thisgraph++)
   readings[thisgraph][thisReading] = 0;
   */

  // Create a new file in the sketch directory
  SimpleTableNew();
  graphitnao = false;
}

void draw () {

  // everything happens in the serialEvent()
}

void keyPressed() {
  if (key == '\n') {
    if (graphitnao == false) {
      graphitnao =true;
    } else
    {
      graphitnao = false;
    }
  }
}


void serialEvent (Serial myPort) {
//  if (graphitnao == false)
//  {
//    println("not graphing");
//    return;
//  }
  // get the ASCII string:
  String inString = myPort.readStringUntil('\n');
  boolean graph = false;
  int yoffset = height/numgraph;
```

```
boolean graphbypass=false;
if (inString != null) {
  // trim off any whitespace:
  inString = trim(inString);

  //this is to avoid problems with short strings running through inString.charAt(1)
  if (inString.length()<2) {
    println("ERROR: SHORT STRING. RETURNING");
    return;
  }

  switch(inString.charAt(0))
  {
  case '$':
    print("$");
    yoffset = yoffset;
    graphbypass=true;
    break;
  case '^':
    print("g");
    yoffset = yoffset;
    graphbypass=false;
    break;
  default:
    println(inString);
    //Non formated data
    // convert to an int and map to the screen height:
    return;
  }

  char swit = inString.charAt(1);
  //discard which graph we are using
  inString = split(inString, ' ')[1];
  // convert to an int and map to the screen height:
  float inByte = float(inString);


  //switch between x,y,z graphs
  switch(swit)
  {
  case 'x':
    print("x");
    stroke(255, 0, 0);
    yoffset = 0;
    println(inByte);
    data[0] = inByte;
    break;
  case 'y':
    print("y");
    stroke(0, 255, 0);
    yoffset = yoffset;
    println(inByte);
    data[1] = inByte;
    break;
  case 'z':
    print("z");
    graph = true;
```

```
      stroke(0, 0, 255);
      yoffset = -yoffset;
      data[2] = inByte;
      println(inByte);
      break;
    case 'q':
      print("q");
      graph = true;
      stroke(0, 0, 255);
      yoffset = -yoffset;
      println(inByte);
      data[3] = inByte;
      data[4] = data[4] +1;
      SimpleTableSave2(data);
      break;
    default:
      print("D");
      println(inString);
      return;
    }

    //    //discard which graph we are using
    //    inString = split(inString, ' ')[1];
    //    // convert to an int and map to the screen height:
    //    float inByte = float(inString);
    inByte = map(inByte, 150, 1023, 0, height);


    //    SimpleTableSave(inByte, timer);


    if (!graphbypass)
    {
      // draw the line:
      //      stroke(127, 34, 255);
      line(xPos, height/2+yoffset, xPos, height/2 - inByte+yoffset);

      // at the edge of the screen, go back to the beginning:
      if (xPos >= width) {
        xPos = 0;
        background(0);
      } else if (graph == true) {
        // increment the horizontal position:
        xPos++;
        timer++;
      }
    }
  }
}



void SimpleTableNew()
{
  table = new Table();
  table.addColumn("x");
  table.addColumn("y");
  table.addColumn("z");
  table.addColumn("q");
  table.addColumn("t");
  saveTable(table, tablelocation);
}

void SimpleTableSave(float x, float t)
{
  table = loadTable("data.csv", "header");
  TableRow newrow = table.addRow();
  newrow.setFloat("x", x);
  newrow.setFloat("t", t);
  saveTable(table, "data/data.csv");
}

void SimpleTableSave2(float dat[])
{
  table = loadTable("data.csv", "header");
  TableRow newrow = table.addRow();
  newrow.setFloat("x", dat[0]);
  newrow.setFloat("y", dat[1]);
  newrow.setFloat("z", dat[2]);
  newrow.setFloat("q", dat[3]);
  newrow.setFloat("t", dat[4]);
  saveTable(table, "data/data.csv");
}
```