Hoverquad

Troy Lee, Drew Shoenbach, Bryant Tang

Device Functionality:

Our device uses four brushless motors to hover above the ground. These four motors are controlled by motor controllers and powered by a power distribution board. The motor controllers are hooked up to a featherboard which sends PWM signals that determine how much throttle needs to be provided to the motor. Then, using the mpu-6050 and its accelerometer and gyroscopes functions, the device is able to be stabilized and moved around at a five degree angle. The hoverquad is able to take in user input from an app on a mobile phone that connects via bluetooth to the feather.

Hardware Components:

- MPU-6050
- Adafruit Feather M0 Bluefruit LE
  - Bought off of Amazon for 30$
- Power Distribution Board
- 4 Motor Controllers
- 4 Brushless DC motors
- 7.4 V, 850 mAh Li-PO battery
- Frame for motors
- Proto-boards
  - From the lab

*Power distribution board, motor controllers, motors, Li-PO battery, and frame generously provided by Shane Langhans.

Design Timeline:

WEEK	DESCRIPTION
1	From our original proposal project submission, we received feedback from one of the ECE TA’s explaining that our “helicopter” project may be too difficult. Since we wanted to keep the main theme of our project, we seeked help from TA Shane Langhans who gave us the idea of doing the “hoverquad.” Week 1 didn’t concern any actual wiring or coding. Once we met up with Shane, we were able to gather all the materials we needed. Most of the components listed were provided by Shane. We bought the

	Adafruit Feather and used the MPU-6050 from the ECE lab kit.
2	Week 2 is when we started to code and solder. We soldered male headers to the feather board and a wire from the ground to the ADD on the MPU 6050. In terms of coding, we wrote a program that correct the values from the MPU into gravitational units. We then calculated the pitch and roll from these values. To test the bluetooth capabilities of the feather, we used the Bluefruit LE app, which is an app used in correlation with the feather (already developed). The building of a platform on top of the hovercraft frame was put into question. We discussed different ways the components could be held together.
3	We finished soldering the wires onto the micro controllers for the motor (mostly to replace wires that were too short). The four propellers of the hovercraft started to function, and all that was left was calibration. Using mathematics, we were able to calculate conversions for the x-y-z coordinates of the hovercraft, the accelerometer, and the gyroscope. In programming the arduino, we were able to use PWM (pulse-width-modulation) to control the motors.
4	At this point, we were focusing on getting the motors to work in accordance to the pitch, yaw, and roll. Everything was working and we basically finished the programming part of the project, with some extra refinement needed. We began to start on building the infrastructure for the components that would sit on top of the hovercraft frame. We also did some soldering to connect all the wires from the motor controllers to the ground and power on the power distribution board.
5	Everything has been soldered and all the wires were either twisted or glued in order to keep organization and create a less messy surface. In the last week, we build the infrastructure using a simple method of super-gluing four proto boards onto the frame. On these four proto boards were the MPU 6050 and the Adafruit Feather. These four boards also created a mechanism of holding the battery and power distribution board, which was located in the middle of the hovercraft. The end result of this week was a finished project. However, while testing the motors to make sure they were all calibrated, one of the motor controllers shorted and we were left with only three working motors. Due to this, the “hoverquad” couldn’t “hover” and we didn’t have enough time nor resources to finish it before the science fair. If everything had gone smoothly, the “hoverquad” would have been perfected.

Software design:

The code uses libraries that come with the feather m0 board, in particular the bluetooth library. In the setup for the program calls an mpu-6050 setup function we defined as below (which uses I2C):

```
void initializeMPU()
{
  Wire.begin();
  Wire.beginTransmission(104);
  Wire.write(107);
  Wire.write(0); // turns on MPU
  Wire.endTransmission(true);
  Wire.beginTransmission(104);
  Wire.write(28);
  Wire.write(2 << 3); //change register AFS_SEL[1:0]
  Wire.endTransmission(true);
  Wire.endTransmission(true);
  Wire.beginTransmission(104);
  Wire.write(29);
  Wire.write(5); //change accel DLPF to 41hz
  Wire.endTransmission(true);
}
```

The rest of the setup waits for the feather to be connected via bluetooth before entering the loop. Once in the loop, four functions are called: readPacket, blueParse, attitudecontroller, and getPitchRoll. The readPacket and blueParse are related to the bluetooth and we only called them every 20 loops (because they run very slowly). The attitudecontroller is only ran everytime a packet from the bluetooth is received (as that is when we're controlling the motors). The getPitchRoll is ran every time since it updates the roll, pitch, and yaw.

blueParse

```
void blueParse()
{
  // printHex(packetbuffer, len);
  // Buttons
  if (packetbuffer[1] == 'B') {
    uint8_t buttnum = packetbuffer[2] - '0';
    boolean pressed = packetbuffer[3] - '0';
    if (pressed)
    {
      if (buttnum == 5)
      {
        direction_command = FORWARD;
        pitch_effort = -1;
        roll_effort = 0;
        yaw_effort = 0;
      }
      else if (buttnum == 7)
      {
        direction_command = LEFT;
        pitch_effort = 0;
        roll_effort = -1;
    yaw_effort = 0;
   }
   else if (buttnum == 6)
   {
    direction_command = BACKWARD;
    pitch_effort = 1;
    roll_effort = 0;
    yaw_effort = 0;
   }
   else if (buttnum == 8)
   {
    direction_command = RIGHT;
    pitch_effort = 0;
    roll_effort = 1;
    yaw_effort = 0;
   }
   else if (buttnum == 1)
   {
    throttle_base = throttle_base + 50;
    NO_THROTTLE = 0;
   }
   else if (buttnum == 2)
   {
    throttle_base = throttle_base - 50;
    if (throttle_base <= 0)
    {
     throttle_base = 0.0;
     NO_THROTTLE = 1;
    }
   }
   else if (buttnum == 3)
   {
    direction_command = TWIST_RIGHT;
   }
   else if (buttnum == 4)
   {
    direction_command = TWIST_LEFT;
   }
  }
  else
  {
   direction_command = NONE;
   yaw_effort = 0;
   roll_effort = 0;
   pitch_effort = 0;
  }
 }
```

attitudeController

```
void attitudeController()
 {
 switch (direction_command)
 {
  case FORWARD:
   roll_set_point = 0.0;
```

```
      pitch_set_point = -angle_movement;
      break;
    case BACKWARD:
      roll_set_point = 0.0;
      pitch_set_point = angle_movement;
      break;
    case RIGHT:
      roll_set_point = angle_movement;
      pitch_set_point = 0.0;
      break;
    case LEFT:
      roll_set_point = -angle_movement;
      pitch_set_point = 0.0;
      break;
    default : break;
  }
  if (!NO_THROTTLE)
  {
    roll_error = roll_set_point - measured_roll;
    roll_error_I += roll_error;
    pitch_error = pitch_set_point - measured_pitch;
    pitch_error_I += pitch_error;
    yaw_error = yaw_set_point - measured_yaw_rate;
    yaw_error_I += yaw_error;
  }
  else
  {
    yaw_error_I = 0;
    roll_error_I = 0;
    pitch_error_I = 0;
  }
  float yaw_effort = 0;//yaw_gain_P * yaw_error + yaw_gain_I * yaw_error_I;
  float pitch_effort = pitch_gain_P * pitch_error + pitch_gain_I * pitch_error_I;
  float roll_effort = roll_gain_P * roll_error + roll_gain_I * roll_error_I;
  int m0 = throttle_base + yaw_effort - pitch_effort + roll_effort;
  int m1 = throttle_base - yaw_effort + pitch_effort + roll_effort;
  int m2 = throttle_base + yaw_effort + pitch_effort - roll_effort;
  int m3 = throttle_base - yaw_effort - pitch_effort - roll_effort;
  if (NO_THROTTLE)
  {
    pwmWrite(10, 0);
    pwmWrite(11, 0);
    pwmWrite(12, 0);
    pwmWrite(13, 0);
  }
  else
  {
    pwmWrite(10, m0 + m0_min);
    pwmWrite(11, m1 + m1_min);
    pwmWrite(12, m2 + m2_min);
    pwmWrite(13, m3 + m3_min);
  }
  }
```

getPitchRoll

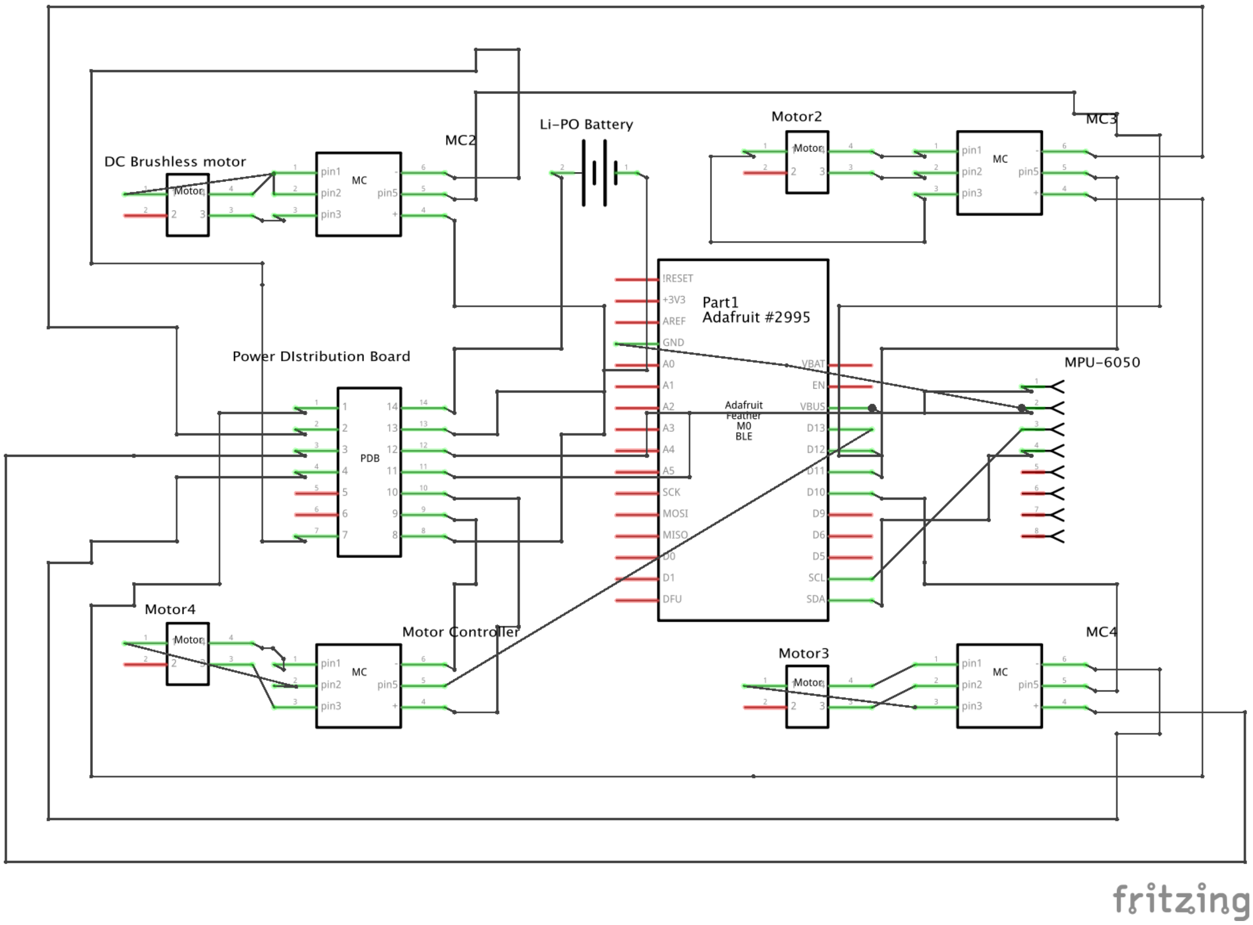
```
void getPitchRoll()
{
  Wire.beginTransmission(104);
  Wire.write(59);
  Wire.endTransmission(false);
  Wire.requestFrom(104, 12, true);
  int16_t ax = (Wire.read() << 8) | Wire.read();
  int16_t ay = (Wire.read() << 8) | Wire.read();
  int16_t az = (Wire.read() << 8) | Wire.read();
  int16_t gyrox = (Wire.read() << 8) | Wire.read();
  int16_t gyroy = (Wire.read() << 8) | Wire.read();
  int16_t gyroz = (Wire.read() << 8) | Wire.read();
  gx = (ax-ax_offset) / accel_scale;
  gy = -(ay-ay_offset) / accel_scale;
  gz = -(az-az_offset) / accel_scale;
  measured_pitch = atan2(gx, sqrt(gz * gz + gy * gy));
  measured_roll = atan2(-gy, -gz);
  measured_yaw_rate = -(gyroz / gyro_scale - gyroz_offset);
}
```

pwmWrite

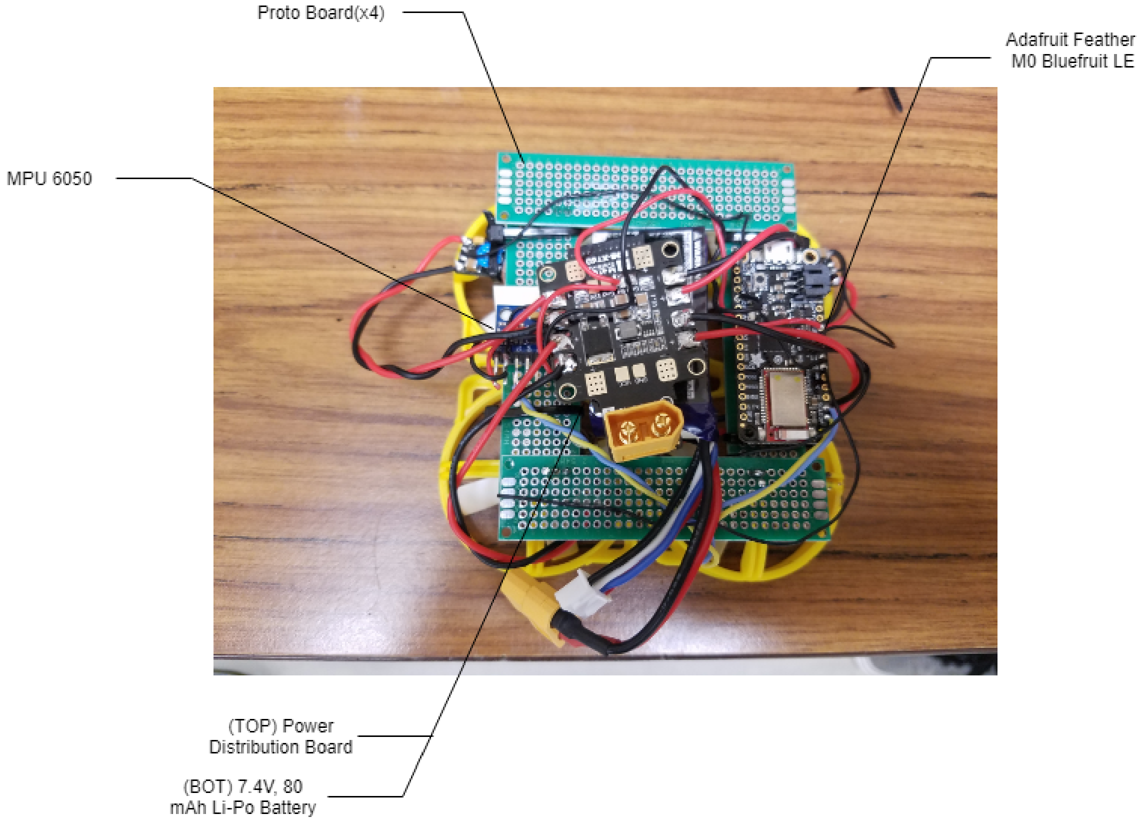
To control the motors we needed to use pwm, so we wrote a pwmWrite function to control the motors.

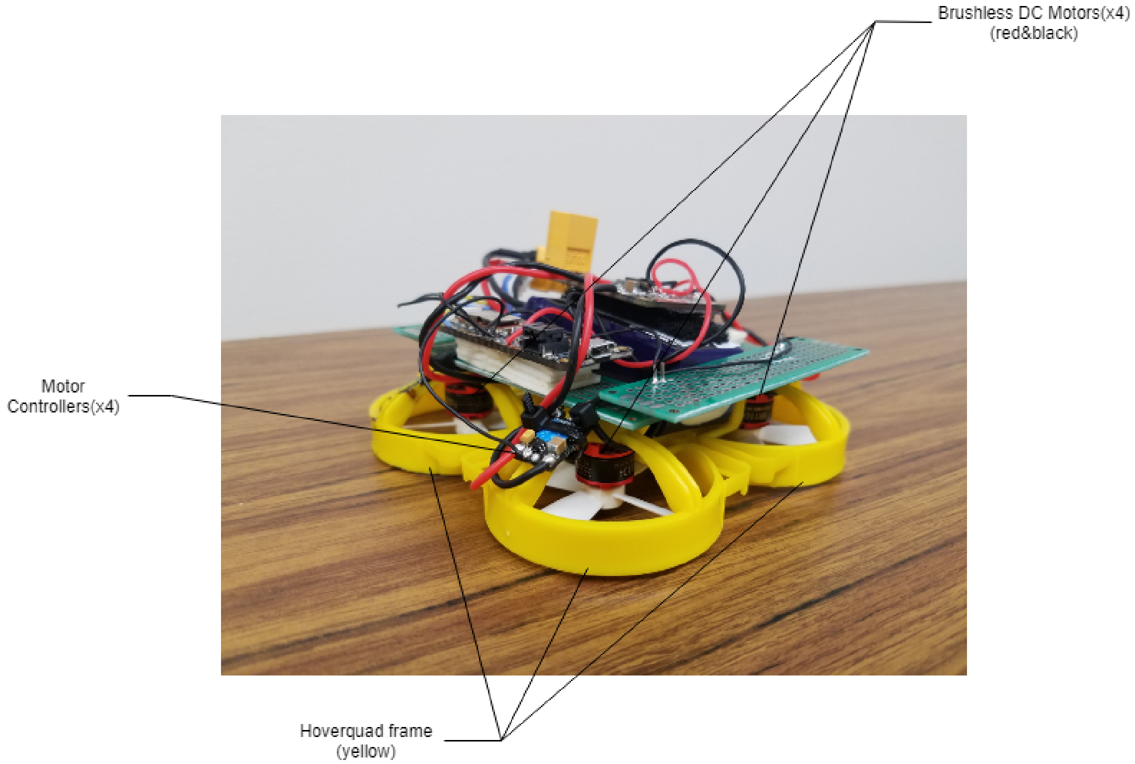
```
void pwmWrite(int pin, int throttle)
{
  if ( throttle > 1000 || throttle < 0)
  {
    return;
  }
  float pulseUS_f = (float)throttle   *  (maxThrottleUS - 1000) / 1000.0 + 1000.0;
  int pulseCounts = (int)(pulseUS_f*countsPerUS);
  if ( pin == 10)
  {
    REG_TCC0_CC2 = pulseCounts;
  }
  else if (pin == 11)
  {
    REG_TCC2_CC0 = pulseCounts;
  }
  else if (pin == 12)
  {
    REG_TCC0_CC3 = pulseCounts;
  }
  else if (pin == 13)
  {
    REG_TCC2_CC1 = pulseCounts;
  }
}
```

Circuit Schematics



Circuit Prototype


Proto Board(x4)
Adafruit Feather M0 Bluefruit LE
MPU 6050
(TOP) Power Distribution Board
(BOT) 7.4V, 80 mAh Li-Po Battery



Testing

One of the major tests we performed was regarding the calibration of our motors. Each motor responded to a different PWM signal, this meant that some required less throttle to turn on. Since we didn’t know this at first we thought either some of the motor controllers didn’t work or the motor themselves didn’t work. The first test we did was to go around to each motor and provide throttle to them. From there we determined which controllers were working and than placed them on the controller motor combos that weren’t working. We were able to determine that all the motors worked, so we thought that some of the controllers didn’t work. To test this, we tried higher throttles on the controller that didn’t work and discovered that they did work. Thus, we learned that each controller had a different throttle amount to start, which required us to calibrate each motor accordingly.