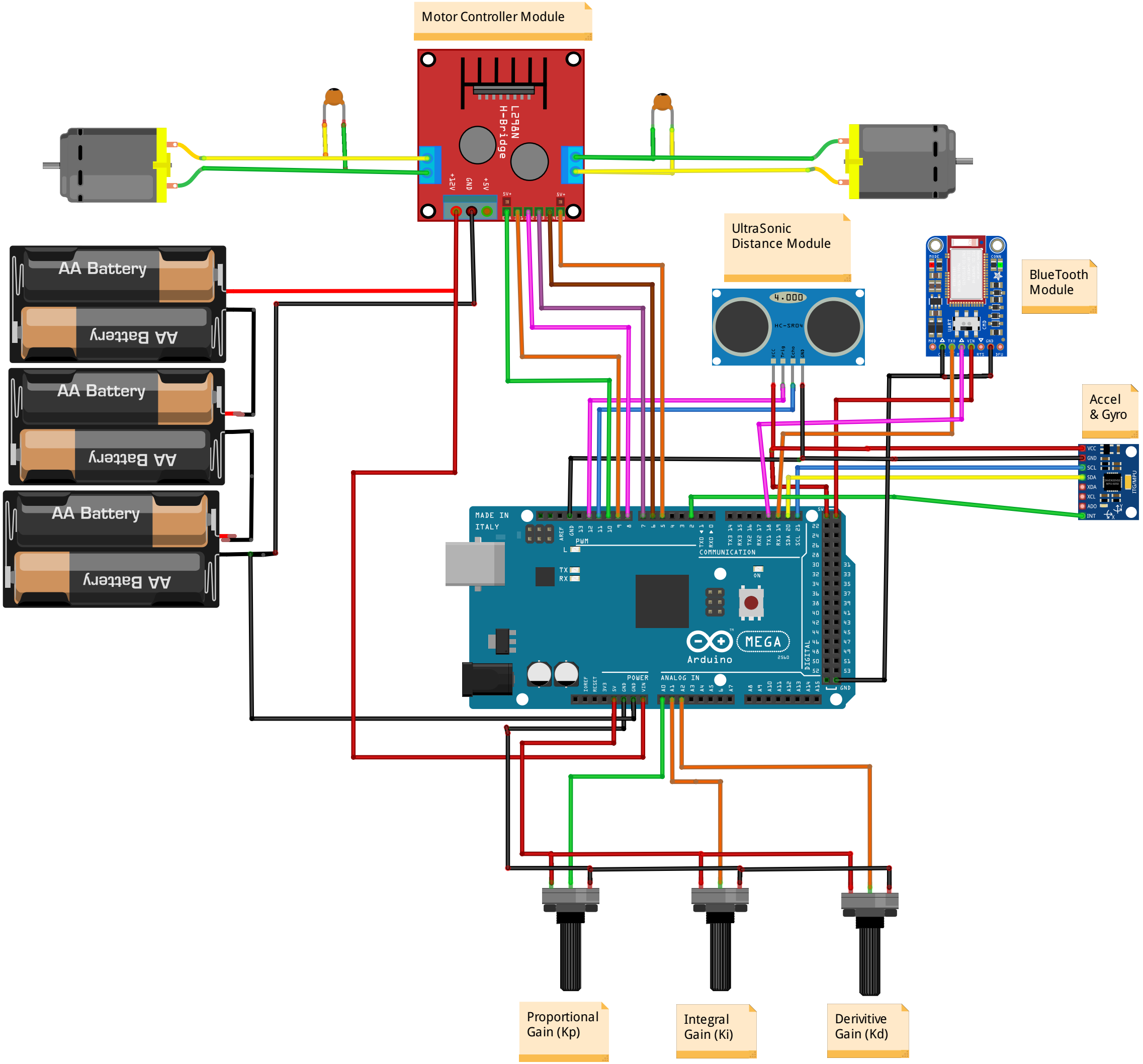
Motor Controller Module
L298N
H-Bridge
+12V
GND
+5V
5V+
AA Battery
AA Battery
AA Battery
AA Battery
UltraSonic
Distance Module
4.000
HC-SR04
Vcc
Trig
Echo
Gnd
BlueTooth
Module
MODE
CONN
UART
CMD
MOD
TXD
VIN
GND
RTS
DFU
Accel
& Gyro
VCC
GND
SCL
SDA
XDA
XCL
ADO
INT
ITG/MPU
MADE IN
ITALY
PWM
COMMUNICATION
TX
RX
ON
MEGA
Arduino
2560
DIGITAL
POWER
ANALOG IN
IOREF
RESET
3V3
5V
GND
GND
VIN
Proportional
Gain (Kp)
Integral
Gain (Ki)
Derivitive
Gain (Kd)

```
/*
************************************
*  Self_Balance Robot              *
*  ==================              *
*   By Roy H Guerra Jr.            *
*   8/13/17                        *
************************************

This code uses an Adafruit BlueTooth Module with an Arduino Mega board to control the
direction and speed of a self-balance robot using the Adafruit iPhone Application
and BLE Library.
Note - Must include BluefruitConfig.h and packetParser.cpp files on seperate tabs on
order for program to function.

Set up and Configure the BLE UART Module for per the Adafruit Manual using Harware Serial
Port 1.

This code uses a LN298N Motor Controller Board with PWM.
Notes:
1) Apply 6V-12V power to L298N Motor Controller
2) Limit PWM high limit to 90% per L298N cut sheet
3) Add 0.1uF capacitors across motor to supress noise

Additional Notes:
 - Roll = x-axis (right is "+", left is "-")
 - Pitch = y-axis (up is "+", down is "-")
 - Adjust Ki, Kp, Kd
```

- Feedback is set to monitor the filtered "pitch" (Y-Axis).
- For self-balance, the setpoint = setpoint + 180
- Add a sceondary loop with an encoder input for speed control (optional)
- Add an Utrasonic Sensor for navigation (optional)
- Can add an LCD display (optional)
- When powering on, calibrate in upward position to get offset values
- Contains a reverse interlock function so turns can not be made (interlock resets in forward direction)

Hardware Required & Connections:

- Adafruit BLE UART Module
- Arduino Mega board
- L298N Motor Controller Board
- MPU6050 6 axis Gyro & Accelerometer Board
- HC-SR04 Utrasonic Module
- 5-9 VDC motors with gearbox
- Rotory encoders are optional (need to modify code if used)
- MISC (solder, proto board, case, etc.)

WiFi Board Modifications:
---
1) Set jumper on Dragino Wi-Fi Board to (VCC-5V)
2) See Instruction manual for setup and operation
3) Can program and troubleshoot "Over The Air", no USB cable required

Motor One Control

```
-----------------
L298N enA = Arduino pin 10
L298N in1 = Arduino pin 9
L298N in2 = Arduino pin 8

Motor Two Control
-----------------
L298N enB = Arduino pin 5
L298N in3 = Arduino pin 7
L298N in4 = Arduino pin 6

L298N Power
-----------
L298N (+) = 6-12 volts
L298N (-) = Gnd (also connect to Gnd on Arduino
L298Nn(5V out) = Connect to +5V on Arduino

L298N Motor Connections
-----------------------
L298N (Out 1) = Motor 1 (+)
L298N (Out 2) = Motor 1 (-)
L298N (Out 3) = Motor 2 (+)
L298N (Out 4) = Motor 2 (-)

MPU6050 Connections
-------------------
VCC = 5 volts Arduino
```

```
Gnd = Gnd Arduino
SDA = #20 Arduino Mega
SCL = #21 Arduino Mega
INT = #2 Arduino Mega

HC-SR04 Utrasonic Module Connections
-----------------------------------
VCC = 5 volts Arduino
Gnd = Gnd Arduino
Trigger Pin = Arduino pin 12
Echo Pin = Arduino pin 11

Adafruit BLE UART Module Connections
-----------------------------------
VCC = 5 volts Arduino
Gnd = Gnd Arduino
CTS = Gnd Arduino
TXO = Arduino Mega RX1 (pin 19)
RX1 = Arduino Mega TX1 (pin 18)
Note - Set DIP Switch on BLE Module to "CMD"

Note - Important, Do not use delays more than 1mS total in this program with the current
interupt
*/

// Libraries
// ---------
```

```
#include <PID_v1.h>
#include <NewPing.h>
#include<LMotorController.h>
#include "I2Cdev.h"
#include"MPU6050_6Axis_MotionApps20.h"
// Include this library if applicable
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
    #include "Wire.h"
#endif
#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined (_VARIANT_ARDUINO_ZERO_)
  #include <SoftwareSerial.h>
#endif
#include"Adafruit_BLE.h"
#include"Adafruit_BluefruitLE_SPI.h"
#include"Adafruit_BluefruitLE_UART.h"
#include"BluefruitConfig.h"

// For MANUAL_TUNING, the following should be set to 1(can always change back once tuned)
//=============================================================================
#define LOG_INPUT 1
#define MANUAL_TUNING 1
#define LOG_PID_CONSTANTS 1
#define MOVE_BACK_FORTH 0
```

```
// Motor Controller Setup
//======================
// Motor One
//----------
#define ENA 10
#define IN1 9
#define IN2 8
// Motor Two
//----------
#define ENB 5
#define IN3 7
#define IN4 6

// HC-SR04 Utrasonic Module Setup
//-------------------------------
#define TRIGGER_PIN 12
#define ECHO_PIN 11
#define MAX_DISTANCE 60

// Setup a differential to compensate for motor non uniformities
//=============================================================
#define MotorSpeedAccelerationLeft 0.90   // 1 is max, this is the motor acceleration
constant; 90% of Duty Cycle
#define MotorSpeedAccelerationRight 0.90  // 1 is max,this is the motor acceleration
constant; 90% of Duty Cycle

// BLE Setup
```

```
// =========
#define FACTORYRESET_ENABLE         0 // Start at 1 (for enable) then change to 0
(disable)
#define MINIMUM_FIRMWARE_VERSION    "0.6.6"
#define MODE_LED_BEHAVIOUR          "MODE"

#define BLUEFRUIT_HWSERIAL_NAME     Serial1  // Comment out if using software Serial

// Create the bluefruit object, either software serial...uncomment these lines

//SoftwareSerial bluefruitSS = SoftwareSerial(BLUEFRUIT_SWUART_TXD_PIN,
BLUEFRUIT_SWUART_RXD_PIN);

//Adafruit_BluefruitLE_UARTble(bluefruitSS,BLUEFRUIT_UART_MODE_PIN,
BLUEFRUIT_UART_CTS_PIN, BLUEFRUIT_UART_RTS_PIN);

/* ...or hardware serial, which does not need the RTS/CTS pins. Uncomment this line */
 Adafruit_BluefruitLE_UARTble(BLUEFRUIT_HWSERIAL_NAME,BLUEFRUIT_UART_MODE_PIN);

/* ...hardware SPI, using SCK/MOSI/MISO hardware SPI pins and then user selected
CS/IRQ/RST */
//Adafruit_BluefruitLE_SPIble(BLUEFRUIT_SPI_CS,BLUEFRUIT_SPI_IRQ,BLUEFRUIT_SPI_RST);

/* ...software SPI, using SCK/MOSI/MISO user-defined SPI pins and then user selected
CS/IRQ/RST */
//Adafruit_BluefruitLE_SPIble(BLUEFRUIT_SPI_SCK,BLUEFRUIT_SPI_MISO,
//                            BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
```

```
//                            BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);

// A small helper
void error(const __FlashStringHelper*err) {
  Serial.println(err);
  while (1);
}

// Function prototypes over in packetparser.cpp
//=========================================
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout);
float parsefloat(uint8_t *buffer);
void printHex(const uint8_t * data, const uint32_t numBytes);

// Packet Buffer
// =============
extern uint8_t packetbuffer[];

// Set up MPU I2C address is 0x68
//===============================
MPU6050 mpu;

// MPU control/status and variables
//=================================
bool dmpReady = false;  // set true if DMP init was successful
uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 =
```

```
error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

// Activate the following Modules for (orientation/motion)
//=====================================================
Quaternion q;           // [w, x, y, z]         quaternion container
VectorFloat gravity;    // [x, y, z]            gravity vector
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity
vector

// Activite for Manual PID Tuning
//==============================
#if MANUAL_TUNING
  double kp , ki, kd;
  double prevKp, prevKi, prevKd;
#endif

// Additional Controller Parameters
//================================
double originalSetpoint = 178.13;
double setpoint = originalSetpoint;
double movingAngleOffset = 0.3;
double input, output;
int moveState=0; //0 = balance; 1 = back; 2 = forth
boolean dir_flag = 0;  // Reverse Interlock flag
```

```
#if MANUAL_TUNING
  PID pid(&input, &output, &setpoint, 0, 0, 0, DIRECT);
#else
  PID pid(&input, &output, &setpoint, 75, 397, 1.6, DIRECT); // Read Serial and enter new
values
#endif

// Set Up Loop Timers
//==================
long time100mS = 0;
long time1S = 0;
long time5S = 0;

volatile bool mpuInterrupt = false;     // indicates whether MPU interrupt pin has gone
high
void dmpDataReady(){
    mpuInterrupt = true;
}

// Set Up Ultrasonic Case
//======================
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

// Motor Controller Case
//======================
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4,
```

```
MotorSpeedAccelerationLeft,MotorSpeedAccelerationRight);

// Initial Setup Routine
//=====================
void setup(){
  // join I2C bus (I2Cdev library doesn't do this automatically)
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        Wire.begin();
        Wire.setClock(400000);
    #elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
        Fastwire::setup(400, true);
     #endif
  Serial.begin(115200); // initialize Serial Communication
  while (!Serial); // wait for Leonardo enumeration, others continue immediately
  // Initialize MPU Device
    // =====================
    Serial.println(F("Initializing I2C devices..."));
    mpu.initialize();

    // Verify Connection
    // =================
    Serial.println(F("Testing device connections..."));
    Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050
connection failed"));

    // Load and Configure DMP
    // ======================
```

```
    Serial.println(F("Initializing DMP..."));
    devStatus = mpu.dmpInitialize();

    // Supply your own gyro offsets here
    // ================================
   // These come from the Calibration Routine form Jeff Rowberg's Library
    mpu.setXAccelOffset(-2718);
    mpu.setYAccelOffset(1102);
    mpu.setZAccelOffset(1830);
    mpu.setXGyroOffset(47);
    mpu.setYGyroOffset(-15);
    mpu.setZGyroOffset(46);

    // If Good, Returns 0
    // ==================
    if (devStatus == 0) {
        // turn on the DMP, now that it's ready
       Serial.println(F("Enabling DMP..."));
        mpu.setDMPEnabled(true);

        // enable Arduino interrupt detection
       Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)...
"));
      attachInterrupt(digitalPinToInterrupt(2), dmpDataReady, RISING);
        mpuIntStatus = mpu.getIntStatus();

        // set our DMP Ready flag so the main loop() function knows it's okay to use it
```

```
        Serial.println(F("DMP ready! Waiting for first interrupt..."));
        dmpReady = true;

        // get expected DMP packet size for later comparison
        packetSize = mpu.dmpGetFIFOPacketSize();

        //Setup PID Controller
        // ====================
        pid.SetMode(AUTOMATIC);
        pid.SetSampleTime(10);
        pid.SetOutputLimits(-255, 255);
    }
    else {
        // ERROR!
        // 1 = initial memory load failed
        // 2 = DMP configuration updates failed
        // (if it's going to break, usually the code will be 1)
        Serial.print(F("DMP Initialization failed (code "));
        Serial.print(devStatus);
        Serial.println(F(")"));
    }
  BLE_Init();  // Goto BlueTooth Initialize Function
}

// Main Program
// ============
void loop(){ // Keep all times short for best control; use loop timers
```

```
    if (!dmpReady) return;

    // wait for MPU interrupt or extra packet(s) available
    while (!mpuInterrupt && fifoCount < packetSize) {

        // No mpu data - performing PID calculations, timer loops, and output to motors

        // Perform PID alogorithim and Move (if required)
        pid.Compute(); // Get Controller output

       motorController.move(output, -50); // Move robot forward/backward at controller
output value or Min Speed (the greater)

        unsigned long currentMillis = millis(); // Store current Timer Value

        if (currentMillis - time100mS >= 100){ // 100mS loop
            loopAt100mS();  // Goto Function
            time100mS = currentMillis;
           }

        if (currentMillis - time1S >= 1000){ // 1 second loop
            loopAt1S();  // Goto Function
            time1S = currentMillis;
             }

        if (currentMillis - time5S >= 5000){ // 5 second loop
            loopAt5S();  // Goto Function
```

```
        time5S = currentMillis;
       }

    Test();  // Goto BlueTooth Function
}

// reset interrupt flag and get INT_STATUS byte
mpuInterrupt = false;
mpuIntStatus = mpu.getIntStatus();

// get current FIFO count
fifoCount = mpu.getFIFOCount();

// check for overflow (this should never happen unless our code is too inefficient)
if ((mpuIntStatus & 0x10) || fifoCount == 1024)
{
    // reset so we can continue cleanly
    mpu.resetFIFO();
   Serial.println(F("FIFO overflow!"));

// otherwise, check for DMP data ready interrupt (this should happen frequently)
}
else if (mpuIntStatus & 0x02){
    // wait for correct available data length, should be a VERY short wait
   while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

    // read a packet from FIFO
```

```
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        // track FIFO count here in case there is > 1 packet available
        // (this lets us immediately read more without waiting for an interrupt)
        fifoCount -= packetSize;

        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);

        #if LOG_INPUT
            Serial.print("ypr\t");
            Serial.print(ypr[0] * 180/M_PI);
            Serial.print("\t");
            Serial.print(ypr[1] * 180/M_PI);
            Serial.print("\t");
            Serial.println(ypr[2] * 180/M_PI);
        #endif

        input = ypr[1] * 180/M_PI + 180; // Calculate controller input
    }
}

void BLE_Init(){ // BlueTooth Initialize Function
  Serial.println(F("Bluefruit App Controller"));
 Serial.println(F("-------------------------"));
```

```
/* Initialise the module */
Serial.print(F("Initialising the Bluefruit LE module: "));

if ( !ble.begin(VERBOSE_MODE) ){
  error(F("Couldn't find Bluefruit, make sure it's in CoMmanD mode & check wiring?"));
}
Serial.println( F("OK!") );

if ( FACTORYRESET_ENABLE ){
  /* Perform a factory reset to make sure everything is in a known state */
  Serial.println(F("Performing a factory reset: "));
  if ( ! ble.factoryReset() ){
    error(F("Couldn't factory reset"));
   }
}

/* Disable command echo from Bluefruit */
ble.echo(false);

Serial.println("Requesting Bluefruit info:");
/* Print Bluefruit information */
ble.info();

Serial.println();

ble.verbose(false);  // debug info is a little annoying after this point!
```

```
 Serial.println(F("******************************"));

  // LED Activity command is only supported from 0.6.6
  if ( ble.isVersionAtLeast(MINIMUM_FIRMWARE_VERSION) ){
    // Change Mode LED Activity
    Serial.println(F("Change LED activity to " MODE_LED_BEHAVIOUR));
   ble.sendCommandCheckOK("AT+HWModeLED=" MODE_LED_BEHAVIOUR);
   }

  // Set Bluefruit to DATA mode
  Serial.println( F("Switching to DATA mode!") );
 ble.setMode(BLUEFRUIT_MODE_DATA);

 Serial.println(F("******************************"));
}

void Test(){ // BlueTooth Function
     // BLE AP Controller Buttons
  // -------------------------
  // Wait for new Blue Tooth Serial Data to arrive
  uint8_t len = readPacket(&ble, BLE_READPACKET_TIMEOUT);
  if (len == 0){
    return;
    }
   BLE_1(); // Goto BlueTooth Control Pad Functions
}
```

```
void BLE_1(){ // BlueTooth Control Pad Functions (depending on motor wire connections
direct may be backwards)
  if (packetbuffer[1] == 'B') {
    uint8_t buttnum = packetbuffer[2] - '0';
    boolean pressed = packetbuffer[3] - '0';

      if ((buttnum == 4)&& (pressed == true)) { // See if dashboard button is pressed
         dir_flag = 0;  // Re-Set Reverse Interlock flag
        motorController.stopMoving();  // Stop both motors
        Serial.println(F("Robot Emergency Stop....."));
        }
      if ((buttnum == 7)&& (pressed == true)) { // See if dashboard button is pressed
         if (dir_flag == 0) {  // Allow turns except for reverse
            motorController.turnLeft(output, 0);  // Turn left, 0 for no kick
            Serial.println(F("Robot Turn Left....."));
          } else {
           Serial.println(F("Right Left Interlock....."));
          }
        }
      if ((buttnum == 8)&& (pressed == true)) { // See if dashboard button is pressed
         if (dir_flag == 0) {  // Allow turns except for reverse
            motorController.turnRight(output, 0);  // Turn Right, 0 for no kick
            Serial.println(F("Robot Turn Right....."));
          } else {
            Serial.println(F("Right Reverse Interlock....."));
          }
        }
```

```
      if ((buttnum == 5)&& (pressed == true)) { // See if dashboard button is pressed
         dir_flag = 0;  // Re-Set Reverse Interlock flag
         motorController.move(output, -50);  // Move Forward
         Serial.println(F("Robot Move Forward....."));
         }
      if ((buttnum == 6)&& (pressed == true)) { // See if dashboard button is pressed
         dir_flag = 1;  // Set Reverse Interlock flag
         motorController.move(output, 50);  // Move Backward
         Serial.println(F("Robot Move Backward....."));
         }
    }
}

void loopAt100mS(){ // Ultrasonic PING Function
  int distanceCm = sonar.ping_cm();  // Compute distance in cm
  if((distanceCm < 45) && (distanceCm != 0)) {  // Turn left if threshold is exceeded
     Serial.println("Distance_cm = " + String(distanceCm));
     motorController.turnLeft(output, 0);  // Turn left (first number is speed @ 90%,
second is "0" for normal speed, no "boost")
  }
}

void loopAt1S() { // Goto function to set tuning parameters
#if MANUAL_TUNING
    setPIDTuningValues(); // Goto Function
#endif
}
```

```
void loopAt5S(){ // Goto function to determine setpoint manually
    #if MOVE_BACK_FORTH
        moveBackForth(); // Goto Function
    #endif
}

// Move Back and Forth Function
//============================
void moveBackForth(){  // Function to determine setpoint manually
    moveState++;
    if (moveState > 2) moveState = 0;

    if (moveState == 0)
      setpoint = originalSetpoint;
    else if (moveState == 1)
      setpoint = originalSetpoint - movingAngleOffset;
    else
      setpoint = originalSetpoint + movingAngleOffset;
}

//PID Tuning Function(3 potentiometers) Function
//==============================================
#if MANUAL_TUNING
void setPIDTuningValues(){  // function to set tuning parameters
    readPIDTuningValues();  // Goto Potentiometer Read Function
    if (kp != prevKp || ki != prevKi || kd != prevKd) {
```

```
#if LOG_PID_CONSTANTS
        Serial.println("Kp Value = " + String(kp));
        Serial.println("Ki Value = " + String(ki));
        Serial.println("Kd Value = " + String(kd));
        Serial.println("Input = " + String(input));
        Serial.println("0utput = " + String(output));
        Serial.println("Setpoint = " + String(setpoint));
#endif
        pid.SetTunings(kp, ki, kd);
        prevKp = kp; prevKi = ki; prevKd = kd;
    }
}

// Function to Read PID Potentiometers
//====================================
void readPIDTuningValues() { // Potentiometer Read Function
    int potKp = analogRead(A0);
    int potKi = analogRead(A1);
    int potKd = analogRead(A2);
    // Scale potentiometer values to the following gains
   // ------------------------------------------------
    kp = map(potKp, 0, 1023, 25000, 0) / 100.0; //(0 - 250)
    ki = map(potKi, 0, 1023, 100000, 0) / 100.0; //(0 - 1000)
    kd = map(potKd, 0, 1023, 500, 0) / 100.0; //(0 - 5)
}
#endif
```

```
#include <string.h>
#include <Arduino.h>
#include <SPI.h>
#if not defined (_VARIANT_ARDUINO_DUE_X_) && not defined (_VARIANT_ARDUINO_ZERO_)
  #include <SoftwareSerial.h>
#endif

#include "Adafruit_BLE.h"
#include"Adafruit_BluefruitLE_SPI.h"
#include"Adafruit_BluefruitLE_UART.h"


#define PACKET_ACC_LEN                  (15)
#define PACKET_GYRO_LEN                 (15)
#define PACKET_MAG_LEN                  (15)
#define PACKET_QUAT_LEN                 (19)
#define PACKET_BUTTON_LEN               (5)
#define PACKET_COLOR_LEN                (6)
#define PACKET_LOCATION_LEN             (15)

//    READ_BUFSIZE            Size of the read buffer for incoming packets
#define READ_BUFSIZE                    (20)


/* Buffer to hold incoming characters */
uint8_tpacketbuffer[READ_BUFSIZE+1];
```

```
/**************************************************************************/
/*!
    @brief  Casts the four bytes at the specified address to a float
*/
/**************************************************************************/
float parsefloat(uint8_t *buffer)
{
  float f = ((float *)buffer)[0];
  return f;
}

/**************************************************************************/
/*!
    @brief  Prints a hexadecimal value in plain characters
    @param  data      Pointer to the byte data
    @param  numBytes  Data length in bytes
*/
/**************************************************************************/
void printHex(const uint8_t * data, const uint32_t numBytes)
{
  uint32_t szPos;
  for (szPos=0; szPos < numBytes; szPos++)
  {
    Serial.print(F("0x"));
    // Append leading 0 for small values
    if (data[szPos] <= 0xF)
    {
```

```
      Serial.print(F("0"));
      Serial.print(data[szPos] & 0xf, HEX);
    }
    else
    {
      Serial.print(data[szPos] & 0xff, HEX);
    }
    // Add a trailing space if appropriate
    if ((numBytes > 1) && (szPos != numBytes - 1))
    {
      Serial.print(F(" "));
    }
  }
  Serial.println();
}

/**************************************************************************/
/*!
    @brief  Waits for incoming data and parses it
*/
/**************************************************************************/
uint8_t readPacket(Adafruit_BLE *ble, uint16_t timeout) {
  uint16_t origtimeout = timeout, replyidx = 0;

  memset(packetbuffer, 0, READ_BUFSIZE);

 // while (timeout--) {  // Original statement
```

```
if (timeout > 0){ // Roy Added
  timeout--; // Roy added
  //if (replyidx >= 20) break; // Original statement
  if (replyidx >= 20) return; // Roy added
  //if ((packetbuffer[1] == 'A') && (replyidx == PACKET_ACC_LEN))
    //break;
  //if ((packetbuffer[1] == 'G') && (replyidx == PACKET_GYRO_LEN))
    //break;
  //if ((packetbuffer[1] == 'M') && (replyidx == PACKET_MAG_LEN))
    //break;
  //if ((packetbuffer[1] == 'Q') && (replyidx == PACKET_QUAT_LEN))
    //break;
  if ((packetbuffer[1] == 'B') && (replyidx == PACKET_BUTTON_LEN))
    //break;  // Original stsement
    return; // Roy added
 // if ((packetbuffer[1] == 'C') && (replyidx == PACKET_COLOR_LEN))
   // break;
  //if ((packetbuffer[1] == 'L') && (replyidx == PACKET_LOCATION_LEN))
   // break;

  while (ble->available()) {
    char c =  ble->read();
    if (c == '!') {
      replyidx = 0;
    }
   packetbuffer[replyidx] = c;
    replyidx++;
```

```
      timeout = origtimeout;
    }

    //if (timeout == 0) break; // Original ststement
   if (timeout == 0) return; //Roy Added
    delay(1);
  }

  packetbuffer[replyidx] = 0;  // null term

  if (!replyidx)  // no data or timeout
    return 0;
  if (packetbuffer[0] != '!')  // doesn't start with '!' packet beginning
    return 0;

  // check checksum!
  uint8_t xsum = 0;
  uint8_t checksum = packetbuffer[replyidx-1];

  for (uint8_t i=0; i<replyidx-1; i++) {
    xsum += packetbuffer[i];
  }
  xsum = ~xsum;

  // Throw an error message if the checksum's don't match*****Roy commented out some
below ***
  if (xsum != checksum)  {
```

```
    //Serial.print("Checksum mismatch in packet : ");
    //printHex(packetbuffer, replyidx+1);
     return 0;
  }

  // checksum passed!
  return replyidx;
}
```

```
// COMMON SETTINGS
//
------------------------------------------------------------------------------
----
// These settings are used in both SW UART, HW UART and SPI mode
//
------------------------------------------------------------------------------
----
#define BUFSIZE                        128   // Size of the read buffer for incoming data
#define VERBOSE_MODE                   true  // If set to 'true' enables debug output
#define BLE_READPACKET_TIMEOUT         50   // Timeout in ms waiting to read a response
(500)


// SOFTWARE UART SETTINGS
//
------------------------------------------------------------------------------
----
// The following macros declare the pins that will be used for 'SW' serial.
// You should use this option if you are connecting the UART Friend to an UNO
//
------------------------------------------------------------------------------
----
//#define BLUEFRUIT_SWUART_RXD_PIN      9    // Required for software serial!
//#define BLUEFRUIT_SWUART_TXD_PIN      10   // Required for software serial!
//#define BLUEFRUIT_UART_CTS_PIN        11   // Required for software serial!( Uno = 11)
//#define BLUEFRUIT_UART_RTS_PIN        -1   // Optional, set to -1 if unused (Uno = -1)
```

```
// HARDWARE UART SETTINGS
//
-----------------------------------------------------------------------------
----
// The following macros declare the HW serial port you are using. Uncomment
// this line if you are connecting the BLE to Leonardo/Micro or Flora
//
-----------------------------------------------------------------------------
----
#ifdef Serial1    // this makes it not complain on compilation if there's no Serial1
  #define BLUEFRUIT_HWSERIAL_NAME     Serial1
#endif


// SHARED UART SETTINGS
//
-----------------------------------------------------------------------------
----
// The following sets the optional Mode pin, its recommended but not required
//
-----------------------------------------------------------------------------
----
#define BLUEFRUIT_UART_MODE_PIN       -1    // Set to -1 if unused else set to 12
```

```
// SHARED SPI SETTINGS
//
-----------------------------------------------------------------------------
----
// The following macros declare the pins to use for HW and SW SPI communication.
// SCK, MISO and MOSI should be connected to the HW SPI pins on the Uno when
// using HW SPI.  This should be used with nRF51822 based Bluefruit LE modules
// that use SPI (Bluefruit LE SPI Friend).
//
-----------------------------------------------------------------------------
----
//#define BLUEFRUIT_SPI_CS               8
//#define BLUEFRUIT_SPI_IRQ              7
//#define BLUEFRUIT_SPI_RST              6    // Optional but recommended, set to -1 if
unused otherwise 6

// SOFTWARE SPI SETTINGS
//
-----------------------------------------------------------------------------
----
// The following macros declare the pins to use for SW SPI communication.
// This should be used with nRF51822 based Bluefruit LE modules that use SPI
// (Bluefruit LE SPI Friend).
//
-----------------------------------------------------------------------------
----
//#define BLUEFRUIT_SPI_SCK              13
```

```
//#define BLUEFRUIT_SPI_MISO            12
//#define BLUEFRUIT_SPI_MOSI            11
```