D4_Yangtze software

Control software:

```
/*
MPU6050 - Arduino
VCC - 3.3V
GND - GND
SDA - 2
SCL - 3
*////////////////////////////////////////////////////////////////////////////////////
//Include I2C library
#include <Wire.h>
#include <Servo.h>
//Declaring some global variables
int gx, gy, gz;
long ax, ay, az, acc_magnitude;
int temp;
long gx_cal, gy_cal, gz_cal;
long loop_timer;
float angle_pitch, angle_roll;
int angle_pitch_buffer, angle_roll_buffer;
boolean set_angle;
float angle_roll_acc, angle_pitch_acc;
float filtered_pitch, filtered_roll;
float kpx, kix, kdx, servo_out_x, measured_x, setpoint_x, prev_error_x, p_error_x, i_error_x,
d_error_x;
float kpy, kiy, kdy, servo_out_y, measured_y, setpoint_y, prev_error_y, p_error_y, i_error_y,
d_error_y;
int Throttle,Roll, Pitch, Yaw;
#define servoPin 11
Servo servo; //Initialize 'servo' as a servo
////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
void setup() {
kpx = 0.7;
kix = 0.07;
kdx = 0.5;
setpoint_x = 0;
kpy = 0.7;
kiy = 0.01;
kdy = 0.2;
setpoint_y = 0;
pinMode(servoPin, OUTPUT); //Set pin 5 as output
servo.attach(servoPin); //Attach 'pitch servo' to pin 5
Wire.begin(); //start I2C. No address means master
Serial.begin(57600); //Serial for debugging
```

```
setup_IMU_registers(); //Setup the registers of the MPU-6050 (500dfs / +/-8g) and start the gyro
calibrate_MPU6050(); //Run calibration procedure for gyro offsets
loop_timer = micros(); //Reset the loop timer. To be used to keep a constant loop-time. Replacement for interrupts in this situation
}
void loop(){
read_mpu6050(); //Read raw acc and gyro data from MPU-6050
gx -= gx_cal; //Subtract calibrated offsets from read gyro data
gy -= gy_cal;
gz -= gz_cal;
//Gyro angle calculations
// 0.000061068 = (1 / 250Hz)/ 65.5
angle_pitch += gx * 0.000061068; //Calculate the traveled pitch angle and add this to the angle_pitch variable
angle_roll += gy * 0.000061068; //Calculate the traveled roll angle and add this to the angle_roll variable
//0.000001065 = 0.000061068 * (3.142(PI) / 180degr) The Arduino sin function is in radians
angle_pitch += angle_roll * sin(gz * 0.000001065); //If the IMU has yawed transfer the roll angle to the pitch angel
angle_roll -= angle_pitch * sin(gz * 0.000001065); //If the IMU has yawed transfer the pitch angle to the roll angel
//Accelerometer angle calculations
acc_magnitude = sqrt((ax*ax)+(ay*ay)+(az*az)); //Calculate the total accelerometer vector
//57.296 = 1 / (3.142 / 180) The Arduino asin function is in radians
angle_pitch_acc = asin((float)ay/acc_magnitude)* 57.296; //Calculate the pitch angle
angle_roll_acc = asin((float)ax/acc_magnitude)* -57.296; //Calculate the roll angle
//Accelerometer calibration offsets
angle_pitch_acc -= 0.0; //Accelerometer calibration value for pitch
angle_roll_acc -= 0.0; //Accelerometer calibration value for roll
if(set_angle){
angle_pitch = angle_pitch * 0.999 + angle_pitch_acc * 0.001; //Correct the drift of the gyro pitch angle with the accelerometer pitch angle
angle_roll = angle_roll * 0.999 + angle_roll_acc * 0.001; //Correct the drift of the gyro roll angle with the accelerometer roll angle
}
else{
angle_pitch = angle_pitch_acc;
angle_roll = angle_roll_acc;
set_angle = true;
}
//To dampen the pitch and roll angles a complementary filter is used
filtered_pitch = (filtered_pitch * 0.9) + (angle_pitch * 0.1); //Take 90% of the output pitch value and add 10% of the raw pitch value
filtered_roll = (filtered_roll * 0.9) + (angle_roll * 0.1); //Take 90% of the output roll value and add 10% of the raw roll value
```

```
//Serial.print(filtered_pitch);
//Serial.print(" ");
//Serial.println(filtered_roll);
//receiveControl();
servo_out_x = pid_x(filtered_roll,15);
if(servo_out_x>180){
servo_out_x = 180;
}
else if(servo_out_x<0){
servo_out_x = 0;
}
//if(micros()>1000000){
servo.write(servo_out_x);
//}
//Serial.println(Throttle);
//Serial.print(" ");
//Serial.print(servo_out_x);
//Serial.print(" ");
//Serial.print(Roll);
//Serial.print(" ");
//Serial.print(Pitch);
//Serial.print(" ");
//Serial.println(Yaw);
//Serial.println(micros() - loop_timer);
//while(micros() - loop_timer < 4000); //Wait until the loop_timer reaches 4000us (250Hz)
before starting the next loop
//loop_timer = micros(); //Reset the loop timer
}
void setup_IMU_registers(){
//Activate MPU6050
Wire.beginTransmission(0x68); //Address the MPU6050
Wire.write(0x6B); //Address the PWR_MGMT_1 register
Wire.write(0x00); //Reset all values to zero
Wire.endTransmission();
//Configure accelerometer for +/-8g
Wire.beginTransmission(0x68);
Wire.write(0x1C); //Send the requested starting register
Wire.write(0x10); //Set the requested starting register
Wire.endTransmission();
//Configure gyro for 500 degrees per second
Wire.beginTransmission(0x68);
Wire.write(0x1B); //Send the requested starting register
Wire.write(0x08); //Set the requested starting register
Wire.endTransmission();
}
void calibrate_MPU6050(){
```

```
//for (int cal_int = 0; cal_int < 10000 ; cal_int ++){ //Take 10000 samples for a good average reading
// read_mpu6050(); //Read from MPU6050
// gx_cal += gx; //Set calibration offsets for x y z
// gy_cal += gy;
// gz_cal += gz;
//}
//gx_cal /= 10000; //Average all the samples
//gy_cal /= 10000;
//gz_cal /= 10000;
gx_cal = -158; //Using previously calculated calibration data from 10,000 samples
gy_cal = 90;
gz_cal = -77;
//Serial.print("gx_cal: ");
//Serial.print(gx_cal);
//Serial.print(" gy_cal: ");
//Serial.print(gy_cal);
//Serial.print(" gz_cal: ");
//Serial.println(gz_cal);
}
void read_mpu6050(){
Wire.beginTransmission(0x68); //Start communicating with MPU6050, using its default address 0x68. Found in datasheet
Wire.write(0x3B); //Tell MPU6050 to start at register 0x3B. Again found from data sheet
Wire.endTransmission(); //Finish writing
Wire.requestFrom(0x68,14); //Request 14 bytes from MPU6050
while(Wire.available() < 14); //Wait until all bytes are received
ax = Wire.read()<<8|Wire.read(); //Add the low and high byte to the ax variable
ay = Wire.read()<<8|Wire.read();
az = Wire.read()<<8|Wire.read();
temp = Wire.read()<<8|Wire.read(); //Add the low and high byte to the temp variable
gx = Wire.read()<<8|Wire.read();
gy = Wire.read()<<8|Wire.read();
gz = Wire.read()<<8|Wire.read();
}
float pid_x(float meas,float set){
prev_error_x = p_error_x;
p_error_x = set-meas;
i_error_x += p_error_x;
d_error_x = p_error_x-prev_error_x;
return (kpx*p_error_x)+(kix*i_error_x)+(kdx*d_error_x);
}
void receiveControl(){
Wire.requestFrom(1,4);
while(Wire.available() < 4);
Throttle = Wire.read();
```

```
Roll = Wire.read();
Pitch = Wire.read();
Yaw = Wire.read();
}
```

Bluetooth code, including currently incomplete code to use IR proximity sensors to avert a crash:

```
#include <Wire.h>
#include <Servo.h>
//Maximum Number of chars per Packet
#define numChars 13
//IR sensors Pins
#define irFrontPin
#define irBackPin
#define irLeftPin
#define irRightPin
//Bluetooth State Pin
#define btStatePin 2
//Battery Voltage Divider Pin
#define batteryVoltPin 15
#define batteryMaxVoltage 8.5
#define batteryMinVoltage 5.5
//Values of the Different Parameters
#define Range 180 //Max value of the control signal (0-Range)
#define midPoint Range/2
#define throttleHover 0 //Value of the throttle at which the quad hovers
//Variables for Control
byte Controls[4]; // Throttle[0] Roll[1] Pitch[2] Yaw[3]
//Serial Variables
char receivedChars[numChars];
char tempChars[numChars];
boolean newData = false;
char oneChar=0;
//Timing Variables
long batteryPrevMillis = -1000*10;
long batteryInterval = 1000*10;
//Bluetooth State Variable
bool btDisconnected=0;
bool btConnected=0;
void setup() {
pinMode(batteryVoltPin,INPUT);
pinMode(btStatePin,INPUT);
pinMode(13,OUTPUT);
pinMode(irFrontPin,INPUT);
pinMode(irBackPin,INPUT);
pinMode(irLeftPin,INPUT);
```

```
pinMode(irRighPin,INPUT);
Wire.begin(1);
Wire.onRequest(requestEvent);
Serial.begin(9600);
}
void loop() {
//unsigned long currentMillis = millis();
//getBatteryVoltage(currentMillis);
checkBluetooth();
if(btDisconnected==0){
recieveSerial();
checknewData();
}
}
//=============================***FUNCTIONS***==================
==========================================
void requestEvent(){ //I2C function to send controls array
Wire.write(Controls,4);
}
void checkBluetooth(){
bool btState = digitalRead(btStatePin);
if(btState){
btConnected = 1;
btDisconnected = 0;
}
else if(btState==0 && btConnected){
btDisconnected = 1;
btConnected = 0;
noSignal();
}
else{
btConnected=0;
btDisconnected=0;
}
}
void recieveSerial() {
static boolean recvInProgress = false;
static byte ndx = 0;
char startMarker = '@';
char endMarker = '!';
char rc;
while (Serial.available() > 0 && newData == false) {
rc = Serial.read();
if (recvInProgress == true) {
if (rc != endMarker) {
receivedChars[ndx] = rc;
```

```
ndx++;
if (ndx >= numChars) {
ndx = numChars - 1;
}
}
else {
receivedChars[ndx] = '\0'; // terminate the string
recvInProgress = false;
ndx = 0;
newData = true;
}
}
else if (rc == startMarker) {
recvInProgress = true;
}
else { //If it gets here then only once char was sent
oneChar=rc;
newData = true;
}
}
}
void checknewData(){
if (newData == true) {
if(oneChar){
processOneChar();
oneChar=0;
}
else{
processPacket();
}
newData = false;
}
}
//**********************SERIAL PROCESSING
FUNCTIONS***********************************************
void processOneChar(){
switch(oneChar){
}
}
void processPacket(){
strcpy(tempChars, receivedChars);
char identifier = receivedChars[0];
switch(identifier){
break;
case 'L': //leftstick
parseXYpad(Controls[3],Controls[0]);
```

```
break;
case 'R': //rightstick
parseXYpad(Controls[1],Controls[2]);
break;
}
}
//************************PACKET PARSING
FUNCTIONS**********************************************************************
void parseXYpad(byte &x,byte &y){ //Format: <*X###Y###>
char * strIndx;
strIndx = &tempChars[1];
strIndx = strtok(strIndx+1, "Y");
x = atoi(strIndx);
strIndx = strtok(NULL, " ");
y = atoi(strIndx);
}
void parseSlider(int &value){ //Format <*###>
char * strIndx;
strIndx = &tempChars[1];
value = atoi(strIndx);
}
//*****************************TIMING
FUNCTIONS*****************************************************
byte getBatteryVoltage(unsigned long &currentMillis){
if(currentMillis-batteryPrevMillis>=batteryInterval){
batteryPrevMillis=currentMillis;
float Vin;
int voltage;
Vin = analogRead(batteryVoltPin);
Vin= (Vin/1023)*10;
Vin= (Vin-batteryMinVoltage)/(batteryMaxVoltage-batteryMinVoltage);
Vin= Vin*100;
voltage = (int)Vin;
if(voltage<0){
voltage=0;
}
if(voltage<=17){
// Serial.print("*D*");
}
//Serial.print("*V"+String(voltage)+'*');
}
}
//*****************************AUTO-
FUNCTIONS**************************************************************************
void noSignal(){
digitalWrite(13,btState);
```

```
Controls[1]=Range/2;
Controls[2]=Range/2;
Controls[3]=Range/2;
if(Controls[0]>(throttleHover*2)){
Controls[0]=Controls[0]-2;
delay(50);
}
if(Controls[0]>throttleHover){
Controls[0]=Controls[0]-1;
delay(50);
}
}
void AutoLand(){
Controls[1]=Range/2;
Controls[2]=Range/2;
Controls[3]=Range/2;
if(Controls[0]>(throttleHover*2)){
Controls[0]=Controls[0]-2;
delay(50);
}
while(Controls[0]>throttleHover){
Controls[0]=Controls[0]-1;
delay[50];
}
}
void processirSensors(){
bool irFront,irBack,irLeft,irRight;
irFront=digitalRead(irFrontPin);
irBack=digitalRead(irBackPin);
irLeft=digitalRead(irLeftPin);
irRight=digitalRead(irRightPin);
if(irFront && irBack && irLeft && irRight){
AutoLand();
}
else if(irFront && irBack && irRight){
evadeLeft();
}
else if(irFront && irBack && irLeft){
evadeRight();
}
else if(irFront && irLeft){
evadeBackRight();
}
else if(irFront && irRight){
evadeBackLeft();
}
```

```
else if(irBack && irLeft){
evadeFrontRight();
}
else if(irBack && irRight){
evadeFrontLeft();
}
else if(irLeft){
evadeRight();
}
else if(irRight){
evadeLeft();
}
}
//*****************************SEMI AUTO CONTROL
FUNCTIONS*********************************************************************************
void processControls(){
bool irFront,irBack,irLeft,irRight;
int upperLimit,lowerLimit;
upperLimit = midPoint+Range/6;
lowerLimit = midPoint-Range/6;
irFront=digitalRead(irFrontPin);
irBack=digitalRead(irBackPin);
irLeft=digitalRead(irLeftPin);
irRight=digitalRead(irRightPin);
if(Controls[1]>upperLimit){
Controls[1] = upperLimit;
}
else if(Controls[1]<lowerLimit){
Controls[1] = lowerLimit;
}
if(Controls[2]>upperLimit){
Controls[2] = upperLimit;
}
else if(Controls[2]<lowerLimit){
Controls[2] = lowerLimit;
}
if(irFront && irBack && irLeft && irRight){
Controls[1] = midPoint;
Controls[2] = midPoint;
Controls[3] = midPoint;
}
else if(irFront && irBack && irRight){
(Controls[1]>midPoint)? Controls[1]=midPoint;
}
else if(irFront && irBack && irLeft){
(Controls[1]<midPoint)? Controls[1]=midPoint;
```

```
}
else if(irFront && irLeft){
(Controls[1]<midPoint)? Controls[1]=midPoint;
(Controls[2]<midPoint)? Controls[2]=midPoint;
}
else if(irFront && irRight){
(Controls[1]>midPoint)? Controls[1]=midPoint;
(Controls[2]<midPoint)? Controls[2]=midPoint;
}
else if(irBack && irLeft){
(Controls[1]<midPoint)? Controls[1]=midPoint;
(Controls[2]>midPoint)? Controls[2]=midPoint;
}
else if(irBack && irRight){
(Controls[1]>midPoint)? Controls[1]=midPoint;
(Controls[2]>midPoint)? Controls[2]=midPoint;
}
else if(irLeft){
(Controls[1]<midPoint)? Controls[1]=midPoint;
}
else if(irRight){
(Controls[1]>midPoint)? Controls[1]=midPoint;
}
}
```

Bluetooth to I2C, Bluetooth part:

```
/* Code Adapted from Robin2 on Arduino Forums
*/
#include <Wire.h>
#include <SoftwareSerial.h>
SoftwareSerial BT(2, 3); // Pin 2 = RX | Pin 3 = TX
#define numChars 32 // Number of bytes sent by BT
char receivedChars[numChars]; // Received string
char tempChars[numChars]; // Temporary string for manipulation
boolean newData = false;
struct XY
{
byte x;
byte y;
} A;
byte QuadData[4]; // [0] = Throttle | [1] = Roll | [2] = Pitch | [3] = Yaw
byte XYZ[3]; // Accelerometer data
void setup()
{
Serial.begin(9600); // Setup serial with PC for debug
```

```
Serial.println("<Arduino is ready>");
Wire.begin(1);
Wire.onRequest(requestEvent); // Register data request event
Wire.onReceive(receiveEvent); // Register data receive event
BT.begin(9600); // Setup BT Serial
int i;
for(i=0; i<4; i++)
{
QuadData[i] = 127;
}
}
void loop()
{
recvWithStartEndMarkers();
showNewData();
}
void recvWithStartEndMarkers()
{
static boolean recvInProgress = false;
static byte ndx = 0;
char startMarker = '<'; // Start marker for BT data
char endMarker = '>'; // End marker for BT data
char rc;
// if( Serial.available() > 0 ) {
while ( BT.available() > 0 && newData == false )
{
rc = BT.read();
if ( recvInProgress == true )
{
if (rc != endMarker)
{
receivedChars[ndx] = rc;
ndx++;
if (ndx >= numChars)
{
ndx = numChars - 1;
}
}
else
{
receivedChars[ndx] = '\0'; // terminate the string
recvInProgress = false;
ndx = 0;
newData = true;
}
}
```

```
else if ( rc == startMarker )
{
recvInProgress = true;
}
}
}
void showNewData()
{
if (newData == true)
{
parseData();
newData = false;
}
}
void parseData()
{
strcpy(tempChars, receivedChars);
switch ( receivedChars[0] )
{
case 'L':
parseXY();
QuadData[0] = A.y; // Throttle
QuadData[3] = A.x; // Roll
break;
case 'R':
parseXY();
QuadData[1] = A.y; // Pitch
QuadData[2] = A.x; // Yaw
break;
}
}
void parseXY() // !X###Y###
{
char* str = &tempChars[1]; // !X {###} Y###
str = strtok(str + 1, "Y");
A.x = atoi(str);
str = strtok(NULL, " "); // !X###Y {###}
A.y = atoi(str);
}
void printData()
{
/*
Serial.print("\nThrottle = ");
Serial.print(QuadData[0]);
Serial.print("\t|\tRoll = ");
Serial.print(QuadData[1]);
```

```
Serial.print("\t|\tPitch = ");
Serial.print(QuadData[2]);
Serial.print("\t|\tYaw = ");
Serial.print(QuadData[3]);
//*/
///*
Serial.print( "\nX: " );
Serial.print( XYZ[0] );
Serial.print( "\tY: " );
Serial.print( XYZ[1] );
Serial.print( "\tZ: " );
Serial.print( XYZ[2] );
//*/
}
void requestEvent()
{
Wire.write(QuadData, 4); // Send 4 byte array
}
void receiveEvent(int n)
{
int i = 0;
while( Wire.available() > 0 ) // Receive 3 bytes
{
XYZ[i] = Wire.read(); // Store in XYZ array
i++;
}
Serial.print("\n\tNEW DATA!!!\t\n");
printData();
sendData();
}
void sendData()
{
BT.print( "\nX: " );
BT.print( XYZ[0] );
BT.print( "\tY: " );
BT.print( XYZ[1] );
BT.print( "\tZ: " );
BT.print( XYZ[2] );
}
```

Bluetooth to I2C, I2C part:

```
#include <Wire.h>
byte QuadData[4]; // [0] = Throttle | [1] = Roll | [2] = Pitch | [3] = Yaw
int xyz[3]; // Accelerometer Data
byte XYZ[3]; // Mapped accelerometer data
```

```
volatile boolean s;
void setup()
{
Serial.begin(9600); // Setup serial with PC for debug
Wire.begin(); // Setup I2C
pinMode(A0, INPUT); // Analog Read Pins
pinMode(A1, INPUT);
pinMode(A2, INPUT);
int i;
for(i=0; i<4; i++)
{
QuadData[i] = 0;
}
// Timer1 setup
cli(); // Disable interupts
TCCR1A = 0; // Set Registers to 0
TCCR1B = 0;
OCR1A = 62499; // Prescaler for 1Hz
TCCR1B |= (1 << WGM12); // Turn on CTC mode and 256 prescaler
TCCR1B |= (1 << CS12) | (0 << CS11) | (0 << CS10);
TIMSK1 |= (1 << OCIE1A); // Enable timer compare interupt
sei(); // Enable interupts
s = false;
pinMode(13, OUTPUT);
}
ISR(TIMER1_COMPA_vect)
{
s = true;
digitalWrite(13, digitalRead(13) ^ 1);
}
void loop()
{
recData();
accelData();
if(s)
{
Serial.print("\n\tSEND DATA!!!\t\n");
printData();
sendData();
s = false;
}
}
void recData()
{
Wire.requestFrom(1,4); // Request 4 bytes from address 1
while(Wire.available() < 4);
```

```
QuadData[0] = Wire.read(); // Throttle
QuadData[1] = Wire.read(); // Roll
QuadData[2] = Wire.read(); // Pitch
QuadData[3] = Wire.read(); // Yaw
}
void sendData()
{
Wire.beginTransmission(1); // Send to address 1
Wire.write(XYZ, 3); // Send 3 bytes
Wire.endTransmission(1);
}
void printData()
{
/*
Serial.print("\nThrottle = ");
Serial.print(QuadData[0]);
Serial.print("\t|\tRoll = ");
Serial.print(QuadData[1]);
Serial.print("\t|\tPitch = ");
Serial.print(QuadData[2]);
Serial.print("\t|\tYaw = ");
Serial.print(QuadData[3]);
//*/
///*
Serial.print( "\nX: " );
Serial.print( XYZ[0] );
Serial.print( "\tY: " );
Serial.print( XYZ[1] );
Serial.print( "\tZ: " );
Serial.print( XYZ[2] );
//*/
}
void accelData()
{
xyz[0] = analogRead(A0); // Read accelerometer
xyz[1] = analogRead(A1);
xyz[2] = analogRead(A2);
XYZ[0] = map( xyz[0], 180, 420, 0, 255); // Map data to byte range
XYZ[1] = map( xyz[1], 180, 420, 0, 255);
XYZ[2] = map( xyz[2], 180, 420, 0, 255);
}
```

I2C transmitter code:

```
#include <Wire.h>
byte x = 0;
```

```
void setup() {
Wire.begin(1); // join i2c bus (address optional for master)
}
void loop() {
Wire.beginTransmission(8); // transmit to device #8
Wire.write("X is: "); // sends five bytes
Wire.write(x); // sends one byte
Wire.endTransmission(); // stop transmitting
x++;
delay(500);
}
```

I2C receiver code:

```
#include <Wire.h>
void setup() {
Wire.begin(8); // join i2c bus with address #8
Wire.onReceive(dataIncoming); // register event
Serial.begin(9600); // start serial for output
}
void loop() {
//delay(100);
}
// function that executes whenever data is received from master
// this function is registered as an event, see setup()
void receiveEvent(int howMany) {
while (1 < Wire.available()) { // loop through all but the last
char c = Wire.read(); // receive byte as a character
Serial.print(c); // print the character
}
int x = Wire.read(); // receive byte as an integer
Serial.println(x); // print the integer
}
```