```
/*****************************************************************************\
 * Descrição: Main para teste final e calibração do controlador PID          *
 * do Drone                                                                  *
 * Autores: Gustavo L. Fernandes e Giácomo A. Dollevedo                      *
 * Data de Criacao:     03jan2021                                            *
 * Ultima Atualização:  07jan2021                                            *
*****************************************************************************/

#include "DroneWiFi.h"
#include "IMU.h"
#include "DroneTimer.h"
#include "FlightControl.h"
#include "ThrottleControl.h"

//Frequencia Base do Timer (1kHz)
#define FREQUENCY 1000
#define TRIGGER_100MS   FREQUENCY/100
#define TRIGGER_CONTROL FREQUENCY/250

//Contador de mensagens enviados por WiFi
int         sent_counter  =   0;
//Alocacao de memoria para mensagem que sera enviada por WiFi
char*       message       = (char*)calloc(1024, sizeof(char));
//Comando recebido via WiFi
String      command       = "";

//Variaveis para contabilizar quanto falta para o trigger de cada tarefa
volatile unsigned char counterWifi = 0;
volatile unsigned char counterControl = 0;

//Variaveis de Trigger de cada tarefa
volatile unsigned char control_trigger = 0;
volatile unsigned char wifi_trigger = 0;
volatile unsigned char imu_trigger = 0;

/*Objeto do tipo Timer para controlar interrupcoes*/
DroneTimer timer;
/*Objeto do tipo WiFI para realizar a transferencia de dados*/
DroneWiFi wifi;
/*Objeto do tipo IMU para leitura e calculo dos parametros sensoriados*/
IMU imu;
/*Objeto do tipo FlightControl para realizar o controle de velocidade de Pitch*/
FlightControl pitchVelPid(0.5f, 0.8f, 0.9f , 'p');
/*Objeto do tipo FlightControl para realizar o controle de velocidade de Roll*/
FlightControl rollVelPid(0.0 ,0.0, 0.0, 'r');
/*Objeto do tipo FlightControl para realizar o controle de velocidade de Roll*/
FlightControl yawVelPid(0.2f ,0.001f, 0.00f, 'y');
/*Objeto do tipo ThrottleControl para realizar o controle dos motores*/
ThrottleControl quadcopter;

mpu rawData;
processedAngles rotations;

int calibrar = 0;
float calx  = 0;
float caly  = 0;
float calz  = 0;

```

```
59
60
61  rotVel setpoints;
62  float addThrottleTemp;
63
64  /*Estado do sistema*/
65  /*0 == modo de testes | 1 == modo joystick*/
66  unsigned char estado_sistema = 0;
67
68  /*Structs com os ganhos Kp, Ki e Kd de Roll e Pitch*/
69  pidGains ganhosRoll;
70  pidGains ganhosPitch;
71
72  /*Struct com os parametros de monitoramento do drone*/
73  droneParams parametrosDrone;
74
75  /*Vetor que contem as velocidades individuais de cada motor*/
76  int* actualVel = (int*)calloc(4, sizeof(int));
77
78  /*Variaveis para coletar o valor de velocidade angular de cada eixo*/
79  float pitchVel = 0;
80  float rollVel = 0;
81  float yawVel = 0;
82
83
84  /*time_count*/
85  /*seta as flags para controle de tempo do programa*/
86  void IRAM_ATTR time_count(){
87
88    counterControl++;
89    counterWifi++;
90
91    //WiFi eh disparado a cada 100ms
92    if(counterWifi >= TRIGGER_100MS){
93      counterWifi = 0;
94      wifi_trigger = 1;
95    }
96
97    //Atualizacao da IMU e rotina de controle a cada 4ms
98    if(imu_trigger == 0){
99        control_trigger = 1;
100       imu_trigger = 1;
101   }
102
103 }
104
105
106
107 /*ROTINA DE INICIALIZACAO DO SISTEMA*/
108 void setup() {
109
110   Serial.begin(115200); //COMENTAR, NAO VAMOS USAR SERIAL NA MAIN!
111   delay(10);
112
113   /*Inicializacao dos motores*/
114   quadcopter.initializeMotors(PINMOTOR1,PINMOTOR2,PINMOTOR3,PINMOTOR4);
115   delay(5000);
116
117   /*Inicializacao do timer*/
```

```
  timer.initTimer(FREQUENCY, time_count);
  imu.initMPU();
  imu.disableYawComp();

//  imu.disableDebug();
//  wifi.disable_debug();
  /*Inicializacao do WiFi*/
  wifi.initWiFi(SSID_GUS, PASS_GUS, IP_GIA, PORT);

  setpoints = wifi.getVel();
  delay(10);

  /*Habilita as interrupcoes do timer*/
  timer.enableTimer();

  /*Calibracao da IMU*/
  while(calibrar < 2000){
    if(imu_trigger){

      imu_trigger = 0;
      imu.readRawMPU();
      rawData = imu.getRawData();

      calx  += rawData.GyX;
      caly  += rawData.GyY;
      calz  += rawData.GyZ;

      calibrar++;
    }
  }
  imu.CalibrateGyro(calx/calibrar, caly/calibrar, calz/calibrar);

  pitchVelPid.setSetPoint(0);
  rollVelPid.setSetPoint(0);

}



void loop() {

  /*Rotina de execucao das operacoes relacionadas ao WiFi*/
  /*Ocorre a cada 100ms*/
  if(wifi_trigger){
    /*Reseta flag do WiFi*/
    wifi_trigger = 0;

    /*Se eh a primeira mensagem de 5, recebe um comando do servidor*/
    if(sent_counter == 0){
      /*Recebe o comando do Servidor*/
      command = wifi.receiveData();
      /*Realiza a rotina de tratamento deste comando*/
      wifi.processComm((String)command);

      /*Troca a flag do sistema para operar no modo Joystick*/
      if(command == "#J"){
        estado_sistema = 1;
      }

```

```
177       /*Troca a flag do sistema para operar no modo teste*/
178       else if(command == "s"){
179         estado_sistema = 0;
180       }
181
182       /*Aqui ocorrem as alteracoes dos parametros de acordo com o input do usuario*/
183       switch(estado_sistema){
184       /*Atualiza as variaveis de acordo com os comandos de teste*/
185       case 0:
186
187         ganhosRoll  = wifi.getPIDGains('r');
188         ganhosPitch = wifi.getPIDGains('p');
189
190         parametrosDrone = wifi.getParams();
191
192         break;
193
194       /*Atualiza as variaveis de acordo com o Joystick*/
195       case 1:
196         /*Valores dos setpoints pelo joystick*/
197         setpoints = wifi.getVel();
198         break;
199
200       default:
201         break;
202       }
203     }
204
205     switch(estado_sistema){
206       /*Atualiza as variaveis de acordo com os comandos de teste*/
207       case 0:
208         /*Atualizacoes de Kp, Ki e Kd para Pitch*/
209         pitchVelPid.setKp(ganhosPitch.kp);
210         pitchVelPid.setKd(ganhosPitch.kd);
211         pitchVelPid.setKi(ganhosPitch.ki);
212         /*Atualizacoes de Kp, Ki e Kd para Roll*/
213         rollVelPid.setKp(ganhosPitch.kp);
214         rollVelPid.setKd(ganhosPitch.kd);
215         rollVelPid.setKi(ganhosPitch.ki);
216
217         /*Atualiza velocidades dos motores baseado no input do usuario*/
218
 quadcopter.setActualVel(parametrosDrone.M1,parametrosDrone.M2,parametrosDrone.M3,par
ametrosDrone.M4);
219
220         actualVel = quadcopter.getActualVel();
221
222         /*Escreve a mensagem que sera enviada ao Servidor*/
223         sprintf(message, "M1:%d\tM2:%d\tM3:%d\tM4:%d || SetPoint: %0.3f || Erro:
%0.3f \n Vel P: %0.3f Angulo P: %0.3f Kp: %0.3f Kd: %0.3f Ki: %0.3f \0",
actualVel[0], actualVel[1], actualVel[2], actualVel[3], pitchVelPid.getSetPoint(),
pitchVelPid.getPreviousError(), imu.getPitchVel(), imu.getRotations().Pitch,
pitchVelPid.getGains().fkp, pitchVelPid.getGains().fkd, pitchVelPid.getGains().fki);
224
225
226         break;
227
228       /*Atualiza as variaveis de acordo com o Joystick*/
229       case 1:
```

```
        /*Atualiza potencia dos motores baseado no comando do joystick*/
        addThrottleTemp = (((float)(setpoints.throttle)/100)+1)*MOTORTHROTTLE;
        quadcopter.setThrottle(addThrottleTemp);

        /*Atualiza o setpoint de velocidade angular de pitch*/
        pitchVelPid.setSetPoint(setpoints.pitch);

        /*Atualiza o setpoint de velocidade angular de roll*/
        rollVelPid.setSetPoint(setpoints.roll);


        /*Escreve a mensagem que sera enviada ao Servidor*/
        sprintf(message, "SetPoint Roll:\t%0.2f | SetPoint Pitch:\t%0.2f\t Potencias M1:%d\tM2:%d\nAngulo Roll:\t%0.2f | Angulo Pitch:\t%0.2f\tPotencias M3:%d\tM4:%d\0", rollVelPid.getSetPoint(), pitchVelPid.getSetPoint(), actualVel[0], actualVel[1], rotations.Roll, rotations.Pitch,actualVel[2], actualVel[3]);

        break;

      default:
        break;
    }

    /*Envia a mensagem*/
    wifi.sendData(message);

    /*Escreve o caractere finalizador para resetar a mensagem*/
    message[0] = '\0';

    /*Aumenta contador de mensagens consecutivas enviadas*/
    sent_counter++;

    /*Com 5 mensagens enviadas consecutivas, reseta o contador*/
    if(sent_counter == 5){
      sent_counter = 0;
    }
  }

  /*Rotina de execucao de atualizacao da IMU e de controle*/
  /*Ocorre a cada 1ms*/
  if(imu_trigger){
    control_trigger = 0;
    imu_trigger = 0;
    /*Atualiza os dados de inercia do drone*/
    imu.update();

    /*Angulos de Pitch e Roll apos filtro complementar*/
    rotations = imu.getRotations();
    /*Angulos de Pitch e Roll apos filtro complementar*/
    pitchVel = imu.getPitchVel();
    rollVel = imu.getRollVel();
    yawVel = imu.getYawVel();


    /*Executa a rotina de controle que atualiza o sinal de cada eixo baseado nos parametros lidos da IMU*/
    quadcopter.Control(rollVelPid, pitchVelPid, yawVelPid, rollVel, pitchVel, yawVel);

```

```
    /*Le a velocidade atual do drone*/
    actualVel = quadcopter.getActualVel();

    /*Atualiza os parametros de velocidade do drone*/
    parametrosDrone.M1 = actualVel[0];
    parametrosDrone.M2 = actualVel[1];
    parametrosDrone.M3 = actualVel[2];
    parametrosDrone.M4 = actualVel[3];
    wifi.setParams(parametrosDrone);

  }
}

```