```cpp
/* ********************************************************************* */
/* File name:        DroneWiFi.cpp                                       */
/* File description: WiFi socket connection handling implementation  */
/* Author name:      Giacomo Dollevedo, Gustavo Fernandes            */
/* Creation date:    18nov2020                                           */
/* Revision date:    07jan2021                                           */
/* ********************************************************************* */

#include "DroneWiFi.h"


/*
*******************************************************************************
*/
/* Method's name:          initWiFi
  */
/* Description:            WiFi initialization. Connects to a network and server.
  */
/*                         Authenticates.
  */
/*
   */
/* Entry parameters:       char* ssid -> WiFi network ssid
   */
/*                         char* pass -> WiFi network password
   */
/*                         char* hostIP -> Server connection IP
  */
/*                         int port    -> Server Connection port
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*******************************************************************************
*/
void DroneWiFi::initWiFi(char* ssid, char* pass, char* hostIP, int port)
{

  connectWifi(ssid, pass);

  unsigned char flag = 0;

  while(flag == 0){
    flag = connectServer(hostIP, port);
  }

  if(debugging_enabled){
    Serial.println("Connected to Server!");
    Serial.println("Sending Authentication...");
  }
  receiveData();
  delay(100);
  sendData( "OK\0");
  if(debugging_enabled)
    Serial.println("\nAuthentication successful!");

```

```
}

/*
*****************************************************************************
*/
/* Method's name:          sendData
   */
/* Description:            WiFi comm. method to send data. Will mark the first 5
bytes  */
/*                         with the message length for the receiver end.
   */
/*
   */
/* Entry parameters:       WiFiClient _serverCon -> Socket object that represents the
   */
/*                                        connection
   */
/*                         String message -> Message that will be sent through socket
   */
/*                         char* len_c -> Array to enable the first 5 bytes marker
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*****************************************************************************
*/
void DroneWiFi::sendData(String message)       // aceita um registro como parâmetro
{
  int   len = int(message.length());
  unsigned int i = 0;

  sprintf(_len_marker, "%05d", len); // 5 digits

  String envio = _len_marker + message;
  _serverCon.print(envio);
  return;

}


/*
*****************************************************************************
*/
/* Method's name:          receiveData
   */
/* Description:            WiFi comm. method to receive data. Will read the first 5
   */
/*                         incoming bytes and use it to read the incoming stream
   */
/*
   */
/* Entry parameters:       WiFiClient _serverCon -> Socket object that represents the
   */
/*                                        connection
   */
/*
   */
```

```
/* Return parameters:      String -> Message read from the connection
  */
/*
*******************************************************************************
*/
String DroneWiFi::receiveData()
{
  String read_size = "";
  int i = 0;
  int n;
  byte message[249];

  while (!_serverCon.available() || _serverCon.available() <= 4)
    delay(1);
  if (_serverCon.available() > 4) {
    while (i < 5) {
      read_size = read_size + char(_serverCon.read());
      i++;
    }

    n = read_size.toInt();
    i = 0;
    read_size = "";
  }

  while (!_serverCon.available() || _serverCon.available() <= (n - 1))
    delay(1);

  if (_serverCon.available() > 0) {
    while (i < n) {
      read_size = read_size + char(_serverCon.read());
      i++;
    }
    //Serial.println(read_size);
    if(debugging_enabled)
      Serial.println(read_size);
    return read_size;
  }

}

/*
*******************************************************************************
*/
/* Method's name:          setParams
   */
/* Description:            Set internal droneParams variable
   */
/*
   */
/* Entry parameters:       droneParams prm -> struct to be set
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*******************************************************************************
*/
```

```
127 void DroneWiFi::setParams(droneParams prm)
128 {
129
130   _params = prm;
131   return;
132
133
134 }
135
136
137 /*
    ****************************************************************************
    */
138 /* Method's name:          getParams
       */
139 /* Description:            Return the internal drone parameters struct
       */
140 /*
       */
141 /* Entry parameters:       n/a
       */
142 /*
       */
143 /* Return parameters:      droneParams -> internal parameters struct
       */
144 /*
    ****************************************************************************
    */
145 droneParams DroneWiFi::getParams()
146 {
147  return _params;
148 }
149
150
151 /*
    ****************************************************************************
    */
152 /* Method's name:          getPIDGains
       */
153 /* Description:            Return the internal drone PID gains to specific axis
      */
154 /*
       */
155 /* Entry parameters:       unsigned char axis -> desired axis gains
      */
156 /*
       */
157 /* Return parameters:      pidGains -> internal PID gains struct
       */
158 /*
    ****************************************************************************
    */
159  pidGains DroneWiFi::getPIDGains(unsigned char axis)
160  {
161   if(axis == 'r'){
162     return _pidRoll;
163   }
164
165   else if(axis == 'p'){
```

```
    return _pidPitch;
  }

  else
    return _pidYaw;
 }


/*
*****************************************************************************
*/
/* Method's name:          connectWifi
   */
/* Description:            Attempts wifi network connection
  */
/*
   */
/* Entry parameters:       char* ssid -> Network SSID (name)
   */
/*                         char* pass -> Network password
  */
/*
   */
/* Return parameters:      unsigned char -> 0 == failed to connect / 1 == success
  */
/*
*****************************************************************************
*/
unsigned char DroneWiFi::connectWifi(char* ssid, char* pass)
{
  _wifiCon.addAP(ssid, pass); // Network name and password

  if(debugging_enabled){
    Serial.println();
    Serial.println();
    Serial.print("Waiting for WiFi... ");
  }
  while (_wifiCon.run() != WL_CONNECTED) {
    if(debugging_enabled)
      Serial.print(".");
    delay(500);
  }

  if(debugging_enabled)
    Serial.println("\nConnected to WiFi!");

  return 1;

}


/*
*****************************************************************************
*/
/* Method's name:          connectServer
   */
/* Description:            Attempts server connection through WiFi network
   */
```

```
/*
   */
/* Entry parameters:       char* hostIP -> Server connection IP
  */
/*                         int port    -> Server Connection port
   */
/*
   */
/* Return parameters:      unsigned char -> 0 == failed to connect / 1 == success
  */
/*
*************************************************************************************
*/
unsigned char DroneWiFi::connectServer(char* hostIP, int port)
{
  if (!_serverCon.connect(hostIP, port)) {
    if(debugging_enabled){
      Serial.println("Connection failed.");
      Serial.println("Waiting 5 seconds before retrying...");
    }
    delay(5000);
    return 0;
  }

  else{
    return 1;
  }
}

/*
*************************************************************************************
*/
/* Method's name:          processComm
   */
/* Description:            State machine to process server communication commands
  */
/*
   */
/* Entry parameters:       String msg -> server command to be handled
  */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************************
*/
void DroneWiFi::processComm(String msg)
{
  int len = (int)msg.length();
  int i = 0;
  unsigned char j = 0;
  unsigned char k_flag = 1;
  char ch;
  char buffer_k[10];

  /*Se o Joystick esta habilitado, a maquina de estados eh diferente*/
  if(joystick_enabled){
    switch (msg[0]){
```

```
251
252     /*Caso 's' sai do modo de Joystick*/
253     case 's':
254       joystick_enabled = 0;
255       if(debugging_enabled)
256         Serial.println("Joystick Disabled.");
257       break;
258
259     /*Caso '5' reseta os setpoints de velocidade*/
260     case '5':
261       _joystickSetpoints.roll   = 0;
262       _joystickSetpoints.pitch  = 0;
263       break;
264
265     /*Caso '8' diminui velocidade de pitch*/
266     case '8':
267       _joystickSetpoints.pitch  -= VEL_INC;
268       if(_joystickSetpoints.pitch <= -10)
269         _joystickSetpoints.pitch = -10;
270
271       break;
272
273     /*Caso '2' aumenta velocidade de pitch*/
274     case '2':
275       _joystickSetpoints.pitch  += VEL_INC;
276       if(_joystickSetpoints.pitch >= 10)
277         _joystickSetpoints.pitch = 10;
278       break;
279
280     /*Caso '8' aumenta velocidade de roll*/
281     case '6':
282       _joystickSetpoints.roll   += VEL_INC;
283       if(_joystickSetpoints.roll >= 10)
284         _joystickSetpoints.roll = 10;
285
286       break;
287
288     /*Caso '8' diminui velocidade de roll*/
289     case '4':
290       _joystickSetpoints.roll   -= VEL_INC;
291       if(_joystickSetpoints.roll <= -10)
292         _joystickSetpoints.roll = -10;
293       break;
294
295     /*Caso '+' aumenta potencia base dos motores*/
296     case '+':
297       _joystickSetpoints.throttle   += VEL_INC;
298       if(_joystickSetpoints.throttle >= 50)
299         _joystickSetpoints.throttle = 50;
300       break;
301
302     /*Caso '-' diminui potencia base dos motores*/
303     case '-':
304       _joystickSetpoints.throttle   -= VEL_INC;
305       if(_joystickSetpoints.throttle <= 0)
306         _joystickSetpoints.throttle = 0;
307       break;
308
309     case '#':
```

```
      break;

    default:
      if(debugging_enabled)
        Serial.println("Unindentified Command.");
      break;
    }

  }

  else{
    while(i < len){
      ch = msg[i];
      switch(ch){
        case '#':
          i++;
          break;
        case 'K':
          i++;
          break;
        case 'S':
          i++;
          break;
        //#ST1000;1000;1000;1000

        case 'T':
          /*Checa se o comando foi do tipo "Set"*/
          if(msg[i-1] == 'S')
            if(len == 22){
              _params.M1 = (msg[3]-48)*1000 + (msg[4]-48)*100 + (msg[5]-48)*10 + (msg[6]-48);
              _params.M2 = (msg[8]-48)*1000 + (msg[9]-48)*100 + (msg[10]-48)*10 + (msg[11]-48);
              _params.M3 = (msg[13]-48)*1000 + (msg[14]-48)*100 + (msg[15]-48)*10 + (msg[16]-48);
              _params.M4 = (msg[18]-48)*1000 + (msg[19]-48)*100 + (msg[20]-48)*10 + (msg[21]-48);
            }
          i = len;
          break;

        case 'G':
          /*Checa se o comando foi apenas o "GO" do sistema*/
          if(msg[i-1]== '#'){
            if(debugging_enabled)
              Serial.println("Main Loop Started!");
            sendData("Ready to fly!\0");
          }



          /*Checa se o comando foi do tipo "Set"*/
          //#SGaxis;kp;ki;kd
          if(msg[i-1] == 'S'){
            i++;
            /*Caso o eixo escolhido tenha sido Roll*/
            if(msg[i] == 'r'){
              i = i + 2;

```

```
365             while(i < len){
366               buffer_k[j] = msg[i];
367               i++;
368               j++;
369
370               if(msg[i] == ';'){
371                 buffer_k[j] = '\0';
372                 Serial.println(buffer_k);
373
374                 if(k_flag == 1){
375                   _pidRoll.kp = atof(buffer_k);
376                   Serial.println(_pidRoll.kp);
377                   k_flag++;
378                 }
379
380                 else if(k_flag == 2){
381                   _pidRoll.ki = atof(buffer_k);
382                   Serial.println(_pidRoll.kp);
383                   k_flag++;
384                 }
385
386                 j = 0;
387                 i++;
388               }
389             }
390
391             buffer_k[j] = '\0';
392             Serial.println(buffer_k);
393
394             _pidRoll.kd = atof(buffer_k);
395             Serial.println(_pidRoll.kd);
396             k_flag = 1;
397
398
399           }
400           /*Caso o eixo escolhido tenha sido Pitch*/
401           else if(msg[i] == 'p'){
402             i = i + 2;
403
404             while(i < len){
405               buffer_k[j] = msg[i];
406               i++;
407               j++;
408
409               if(msg[i] == ';'){
410                 buffer_k[j] = '\0';
411
412                 if(k_flag == 1){
413                   _pidPitch.kp = atof(buffer_k);
414                   k_flag++;
415                 }
416
417                 else if(k_flag == 2){
418                   _pidPitch.ki = atof(buffer_k);
419                   k_flag++;
420                 }
421                 j = 0;
422                 i++;
423               }
```

```
424             }
425
426             buffer_k[j] = '\0';
427             _pidPitch.kd = atof(buffer_k);
428             k_flag = 1;
429           }
430
431           /*Qualquer outro caso, cai no Yaw*/
432           else{
433             i = i + 2;
434
435             while(i < len){
436               buffer_k[j] = msg[i];
437               i++;
438               j++;
439
440               if(msg[i] == ';'){
441                 buffer_k[j] = '\0';
442
443                 if(k_flag == 1){
444                   _pidYaw.kp = atof(buffer_k);
445                   k_flag++;
446                 }
447                 else if(k_flag == 2){
448                   _pidYaw.ki = atof(buffer_k);
449                   k_flag++;
450                 }
451
452                 j = 0;
453                 i++;
454               }
455             }
456
457             buffer_k[j] = '\0';
458             _pidYaw.kd = atof(buffer_k);
459             k_flag = 1;
460           }
461
462         }
463         i++;
464         break;
465
466       /*NAO COMPLETO!*/
467       /*Alteracao de velocidade*/
468       /*NAO COMPLETO!*/
469       case 'V':
470         /*NAO COMPLETO!*/
471         /*if(len == 7){
472           _params.setPoint = (msg[3]-48)*10 + (msg[4]-48);
473           _params.time = (msg[6]-48);
474         }
475
476         i = len;*/
477         break;
478
479       /*Comando para resetar a potencia dos motores*/
480       case 'R':
481
482         _params.M1 = 1200;
```

```
483           _params.M2 = 1200;
484           _params.M3 = 1200;
485           _params.M4 = 1200;
486 
487           i = len;
488           break;
489 
490         /*Comando para Habilitar Joystick*/
491         case 'J':
492           joystick_enabled = 1;
493           i++;
494           if(debugging_enabled)
495             Serial.println("Joystick Enabled.");
496           break;
497 
498         /*Comandos que nao foram identificados*/
499         default:
500           if(debugging_enabled)
501             Serial.println("Unindentified Command.");
502           i = len;
503       }
504     }
505   }
506 
507   return;
508 }
509 
510 /*
    ********************************************************************************
    */
511 /* Method's name:          getVel
      */
512 /* Description:            Return the internal joystick setpoints struct
       */
513 /*
       */
514 /* Entry parameters:       n/a
       */
515 /*
       */
516 /* Return parameters:      rotVel -> internal joystick setpoints struct
      */
517 /*
    ********************************************************************************
    */
518 rotVel DroneWiFi::getVel(){
519   return _joystickSetpoints;
520 }
521 
522 /*
    ********************************************************************************
    */
523 /* Method's name:          enableDebug
       */
524 /* Description:            Enables Serial Comm Printing
      */
525 /*
       */
```

```
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************
*/
void DroneWiFi::enableDebug(){
  debugging_enabled = 1;
}

/*
*************************************************************************
*/
/* Method's name:          disableDebug
*/
/* Description:            Disables Serial Comm Printing
   */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************
*/
void DroneWiFi::disableDebug(){
  debugging_enabled = 0;
}
```