```
/*************************************************************************
 * Descrição: Arquivo c++ que implementa a interface de controle        *
 * de velocidade dos motores do Drone                                   *
 * Autores: Gustavo L. Fernandes e Giácomo A. Dollevedo                 *
 * Ultima Atualização: 03/01/2021                                       *
*************************************************************************/
#include "ThrottleControl.h"


/* ************************************************************************
*/
/* Nome do metodo:          ThrottleControl
*/
/* Descrição:               Método construtor do objeto, não faz nada além de
*/
/*                          associar 4 objetos do tipo Servo que serão utilizados
*/
/*
*/
/* Parametros de entrada:    Nenhum (Vazio)
*/
/*
*/
/* Parametros de saida: Nenhum (Vazio)
*/
/*
*/
/*
*/
/* ************************************************************************
*/
ThrottleControl::ThrottleControl()
{
  Serial.println("Objeto de Drone Criado!");
  _actualVel = (int*) calloc(3, sizeof(int));
}

/* ************************************************************************
*/
/* Nome do metodo:          initializeMotors
*/
/* Descrição:               Inicia cada um dos motores criados, atrelando a uma
saída*/
/*                          PWM do ESP32 e setando os limites max e min que define
*/
/*                          também a inicialização dos 4 ESC
*/
/*
*/
/* Parametros de entrada: int pinMotor1, int pinMotor2, int pinMotor3, int
pinMotor4*/
/*                         Pinos PWM que serão conectados os sinais de controle
*/
/* Parametros de saida: Nenhum (Vazio)
*/
/*
*/
```

```
/*
*/
/* ************************************************************************
*/

void ThrottleControl::initializeMotors(int pinMotor1, int pinMotor2, int pinMotor3,
int pinMotor4)
{

    _m1.attach(pinMotor1,1000, 2000);
    _m2.attach(pinMotor2,1000, 2000);
    _m3.attach(pinMotor3,1000, 2000);
    _m4.attach(pinMotor4,1000, 2000);

    _m1.write(1000);
    _m2.write(1000);

    _m3.write(1000);
    _m4.write(1000);

     delay(2000);


}

/* ************************************************************************
*/
/* Nome do metodo:         setActualVel
*/
/* Descrição:              Define as velocidades que serão atingidas pelos 4
motores*/
/*
*/
/*
*/
/* Parametros de entrada: int desiredVel1, int desiredVel2, int desiredVel3,
*/
/*                        int desiredVel4 , que são as velocidades desejadas
*/
/*                        aceita valores entre 1000 e 2000 mas limita em 1500
*/
/* Parametros de saida: Nenhum (Vazio)
*/
/*
*/
/*
*/
/* ************************************************************************
*/

void ThrottleControl::setActualVel(int desiredVel1, int desiredVel2, int desiredVel3,
int desiredVel4)
{
//  if((desiredVel1 < 1500) & (desiredVel2 < 1500) & (desiredVel3 < 1500) &
(desiredVel4 < 1500)){
    _m1.write(desiredVel1);
    _m2.write(desiredVel2);
    _m3.write(desiredVel3);
    _m4.write(desiredVel4);
```

```
//  }
}

/* ************************************************************************
*/
/* Nome do metodo:          getActualVel
*/
/* Descrição:               Consulta o sinal pwm definido nos 4 motores
*/
/*
*/
/*
*/
/* Parametros de entrada: Nenhum (Vazio)
*/
/*
*/
/* Parametros de saida:   int* array com os 4 valores de velocidades lidos
*/
/*
*/
/*
*/
/* ************************************************************************
*/


int* ThrottleControl::getActualVel(){

    _actualVel[0] = 1000 + (5.555* _m1.read());
    _actualVel[1] = 1000 + (5.555* _m2.read());
    _actualVel[2] = 1000 + (5.555* _m3.read());
    _actualVel[3] = 1000 + (5.555* _m4.read());

    return _actualVel;
}

/* ************************************************************************
*/
/* Nome do metodo:          gtestMotors
*/
/* Descrição:               Testa se os motores estão funcinoando e respondendo
*/
/*
*/
/*
*/
/* Parametros de entrada: Nenhum (Vazio)
*/
/*
*/
/* Parametros de saida:   boolean  (1 Funcionando) (0 Com problemas)
*/
/*
*/
/*
*/
/* ************************************************************************
*/
```

```
116
117 void ThrottleControl:: testMotors(){
118       _m1.write(1500);
119       delay(5000);
120       _m1.write(1000);
121
122       _m2.write(1500);
123       delay(5000);
124       _m2.write(1000);
125
126       _m3.write(1500);
127       delay(5000);
128       _m3.write(1000);
129
130       _m4.write(1500);
131       delay(5000);
132       _m4.write(1000);
133 }
134
135 /* ************************************************************************
    */
136 /* Nome do metodo:          Control
    */
137 /* Descrição:               Distribuia a velocidade controlada para os 4 motores
    */
138 /*
    */
139 /*
    */
140 /* Parametros de entrada: FlightControl pidRoll, FlightControl pidPitch,
    */
141 /*                        FlightControl pidYaw  , que são os objetos
    */
142 /*                        do controlador implementado para cada eixo
    */
143 /*
    */
144 /* Parametros de saida:  Vazio (Nenhum)
    */
145 /*
    */
146 /*
    */
147 /* ************************************************************************
    */
148
149 void ThrottleControl::Control(FlightControl pidRoll, FlightControl
    pidPitch,FlightControl pidYaw, float rollVel, float pitchVel, float yawVel){
150
151     //Executa a atualização do sinal de controle de cada um dos eixos
152
153     pidRoll.pidVelControl(rollVel);
154     pidPitch.pidVelControl(pitchVel);
155     pidYaw.pidVelControl(yawVel);
156     int desiredVel1, desiredVel2, desiredVel3, desiredVel4;
157
158
159
```

```
160     //Calcula as compensações em cada motor para manter o controle de cada um dos
    eixos de movimentação
161     desiredVel1 = _throttle - pidPitch.getPID_Calculated() -
    pidRoll.getPID_Calculated() - pidYaw.getPID_Calculated();
162     desiredVel2 = _throttle - pidPitch.getPID_Calculated() +
    pidRoll.getPID_Calculated() + pidYaw.getPID_Calculated() ;
163     desiredVel3 = (_throttle +30) + pidPitch.getPID_Calculated() -
    pidRoll.getPID_Calculated() + pidYaw.getPID_Calculated();
164     desiredVel4 = _throttle + pidPitch.getPID_Calculated() +
    pidRoll.getPID_Calculated() - pidYaw.getPID_Calculated();
165
166     //Vamos saturar as velocidades maximas em cada motor
167     if(desiredVel1 > MAXTHROTTLE){
168       desiredVel1 = MAXTHROTTLE;
169     }
170     if(desiredVel2 > MAXTHROTTLE){
171       desiredVel2 = MAXTHROTTLE;
172     }
173     if(desiredVel3 > MAXTHROTTLE){
174       desiredVel3 = MAXTHROTTLE;
175     }
176     if(desiredVel4 > MAXTHROTTLE){
177       desiredVel4 = MAXTHROTTLE;
178     }
179
180     //Seta a nova velocidade necessária para manter a saida controlada
181     setActualVel(desiredVel1, desiredVel2, desiredVel3, desiredVel4);
182
183 }
184
185 /* ************************************************************************
    */
186 /* Nome do metodo:          SingleAxisVelControl
    */
187 /* Descrição:               Distribui  a velocidade controlada para os 2 motores
    */
188 /*                          de modo a contrlar apenas um eixo de movimento
    */
189 /*
    */
190 /* Parametros de entrada: FlightControl pidPitch que é objeto do eixo pitch de
    */
191 /*                         controle            ,
    */
192 /*
    */
193 /*
    */
194 /* Parametros de saida:  Vazio (Nenhum)
    */
195 /*
    */
196 /*
    */
197 /* ************************************************************************
    */
198
199 void ThrottleControl::SingleAxisVelPitchControl(FlightControl pidPitch){
200     int desiredVel1, desiredVel2;
```

```
201
202
203      //Calcula as compensações em cada motor para manter o controle de apenas um dos
eixos de movimentação (pitch)
204     desiredVel1 = _throttle  - pidPitch.getPID_Calculated() ;
205     desiredVel2 = _throttle  + pidPitch.getPID_Calculated() ;
206
207     //Vamos saturar as velocidades maximas em cada motor
208     if(desiredVel1 > MAXTHROTTLE){
209       desiredVel1 = MAXTHROTTLE;
210     }
211     if(desiredVel2 > MAXTHROTTLE){
212       desiredVel2 = MAXTHROTTLE;
213     }
214
215
216     //Seta a nova velocidade necessária para manter a saida controlada
217     setActualVel(desiredVel1, desiredVel2 ,1000, 1000 );
218 }
219
220
221 /* ************************************************************************
*/
222 /* Nome do metodo:         SingleAxisVelControl
*/
223 /* Descrição:              Distribui  a velocidade controlada para os 2 motores
*/
224 /*                         de modo a contrlar apenas um eixo de movimento
*/
225 /*
*/
226 /* Parametros de entrada: FlightControl pidRoll que é objeto do eixo roll de
*/
227 /*                        controle           ,
*/
228 /*
*/
229 /*
*/
230 /* Parametros de saida:  Vazio (Nenhum)
*/
231 /*
*/
232 /*
*/
233 /* ************************************************************************
*/
234
235 void ThrottleControl::SingleAxisVelRollControl(FlightControl pidRoll){
236     int desiredVel1, desiredVel2;
237
238
239      //Calcula as compensações em cada motor para manter o controle de apenas um dos
eixos de movimentação (Roll)
240     desiredVel1 = _throttle  - pidRoll.getPID_Calculated() ;
241     desiredVel2 = _throttle  + pidRoll.getPID_Calculated() ;
242
243     //Vamos saturar as velocidades maximas em cada motor
244     if(desiredVel1 > MAXTHROTTLE){
```

```
245       desiredVel1 = MAXTHROTTLE;
246     }
247     if(desiredVel2 > MAXTHROTTLE){
248       desiredVel2 = MAXTHROTTLE;
249     }
250 
251 
252     //Seta a nova velocidade necessária para manter a saida controlada
253     setActualVel(desiredVel1, desiredVel2 ,1000, 1000 );
254 }
255 
256 
257 
258 
259 
260 /* ************************************************************************
    */
261 /* Nome do metodo:          getThrottle
    */
262 /* Descrição:               Consulta a velocidade (pwm) base atual dos motores
    */
263 /*
    */
264 /*
    */
265 /* Parametros de entrada: Vazio (Nenhum)
    */
266 /*
    */
267 /*
    */
268 /*
    */
269 /* Parametros de saida:  _throttle (int)
    */
270 /*
    */
271 /*
    */
272 /* ************************************************************************
    */
273 int ThrottleControl::getThrottle(){
274   return _throttle;
275 }
276 
277 /* ************************************************************************
    */
278 /* Nome do metodo:          setThrottle
    */
279 /* Descrição:               Define nova velocidade (pwm) base atual dos motores
    */
280 /*
    */
281 /*
    */
282 /* Parametros de entrada: throttleDesired (int)
    */
283 /*
    */
```

```
/*
*/
/*
*/
/* Parametros de saida:  Vazio (Nenhum )
*/
/*
*/
/*
*/
/* ******************************************************************************
*/
void ThrottleControl::setThrottle(int throttleDesired){
  _throttle = throttleDesired;
}

```