```
/* ************************************************************** */
/* File name:        IMU.cpp                                      */
/* File description: MPU-6050 interface implementation file       */
/* Author name:      Giacomo Dollevedo, Gustavo Fernandes         */
/* Creation date:    18nov2020                                    */
/* Revision date:    06jan2021                                    */
/* ************************************************************** */

#include "IMU.h"


/*
**************************************************************************************
*/
/* Method's name:          writeRegMPU
   */
/* Description:            Writes to a MPU-6005 register through I2C bus
   */
/*
   */
/* Entry parameters:       int reg -> Register to write to
   */
/*                         int val -> Value to write
   */
/*
   */
/* Return parameters:      n/a
   */
/*
**************************************************************************************
*/
void IMU::writeRegMPU(int reg, int val)      //aceita um registro e um valor como
parâmetro
{
  Wire.beginTransmission(MPU_ADDR);     // inicia comunicação com endereço do MPU6050
  Wire.write(reg);                      // envia o registro com o qual se deseja
trabalhar
  Wire.write(val);                      // escreve o valor no registro
  Wire.endTransmission(true);           // termina a transmissão
}

/*
**************************************************************************************
*/
/* Method's name:          readRegMPU
  */
/* Description:            Reads from a MPU-6005 register through I2C bus
  */
/*
   */
/* Entry parameters:       unsigned char reg -> Register to read from
  */
/*
   */
/* Return parameters:      unsigned char -> value that was read
  */
/*
**************************************************************************************
```

```
*/
unsigned char IMU::readRegMPU(unsigned char reg)        // aceita um registro como parâmetro
{
  unsigned char _processedData;
  Wire.beginTransmission(MPU_ADDR);     // inicia comunicação com endereço do MPU6050
  Wire.write(reg);                      // envia o registro com o qual se deseja trabalhar
  Wire.endTransmission(false);          // termina transmissão mas continua com I2C aberto (envia STOP e START)
  Wire.requestFrom(MPU_ADDR, 1);        // configura para receber 1 byte do registro escolhido acima
  _processedData = Wire.read();                   // lê o byte e guarda em '_processedData'
  return _processedData;                          //retorna '_processedData'
}

/*
**************************************************************************************
*/
/* Method's name:           findMPU
   */
/* Description:             Check for MPU-6050 address on I2C bus
   */
/*
   */
/* Entry parameters:        n/a
   */
/*
   */
/* Return parameters:       unsigned char -> 0 == not found / 1 == found
  */
/*
**************************************************************************************
*/
unsigned char IMU::findMPU()
{
  Wire.beginTransmission(MPU_ADDR);
  int _processedData = Wire.endTransmission(true);
  unsigned char ucFound = 0;

  if(_processedData == 0)
  {
    if(debbuging_enabled){
      Serial.print("Dispositivo encontrado no endereço: 0x");
      Serial.println(MPU_ADDR, HEX);
    }
    ucFound = 1;
  }
  else
  {
    if(debbuging_enabled)
      Serial.println("Dispositivo não encontrado!");
    ucFound = 0;
  }

  return ucFound;
}

```

```
/*
*************************************************************************************
*/
/* Method's name:          checkMPU
  */
/* Description:            Check MPU-6050 status through I2C bus
   */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      unsigned char -> 0 = not available / 1 = Active / 2 =
Sleep  */
/*
*************************************************************************************
*/
unsigned char IMU::checkMPU()
{

  unsigned char ucCheck = 0;

  if(!findMPU()){
    return ucCheck;
  }

  int _processedData = readRegMPU(WHO_AM_I); // Register 117 – Who Am I - 0x75

  if(_processedData == 104)
  {
    if(debbuging_enabled)
      Serial.println("MPU6050 Dispositivo respondeu OK! (104)");

    _processedData = readRegMPU(PWR_MGMT_1); // Register 107 – Power Management 1-
0x6B

    if(_processedData == 64){
      if(debbuging_enabled)
        Serial.println("MPU6050 em modo SLEEP! (64)");
      ucCheck = 2;
    }
    else{
      if(debbuging_enabled)
        Serial.println("MPU6050 em modo ACTIVE!");
      ucCheck = 1;
    }
  }
  else {
    if(debbuging_enabled)
      Serial.println("Verifique dispositivo - MPU6050 NÃO disponível!");
  }

  return ucCheck;

}
```

```
/*
*************************************************************************************
*/
/* Method's name:          initMPU
   */
/* Description:            Initialize I2C bus and MPU-6050
   */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************************
*/
void IMU::initMPU()
{
  Wire.begin(I2C_SDA, I2C_SCL);
  Wire.setClock(400000);
  setSleepOff();
  setGyroScale();
  setAccelScale();
  checkMPU();
}

/*
*************************************************************************************
*/
/* Method's name:          setSleepOff
   */
/* Description:            Writes to specific register on MPU-6050 to set Active Mode
  */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************************
*/
void IMU::setSleepOff()
{
  writeRegMPU(PWR_MGMT_1, 0); // escreve 0 no registro de gerenciamento de
energia(0x68), colocando o sensor em o modo ACTIVE
}

/*
*************************************************************************************
*/
/* Method's name:          setGyroScale
  */
/* Description:            Set gyroscope scale to +- 250°/s
  */
```

```
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************************
*/
void IMU::setGyroScale()
{
  writeRegMPU(GYRO_CONFIG, 0);
}

/*
*************************************************************************************
*/
/* Method's name:          setAccelScale
   */
/* Description:            Set accelerometer scale to +- 2g
  */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************************
*/
void IMU::setAccelScale()
{
  writeRegMPU(ACCEL_CONFIG, 0);
}

/*
*************************************************************************************
*/
/* Method's name:          readRawMPU
  */
/* Description:            Reads all sensor registers from MPU-6050 through I2C bus
  */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      mpu -> Struct containing raw read values
  */
/*
*************************************************************************************
*/
mpu IMU::readRawMPU()
{
```

```
194
195   int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
196
197   Wire.beginTransmission(MPU_ADDR);       // inicia comunicação com endereço do
MPU6050
198   Wire.write(ACCEL_XOUT);                 // envia o registro com o qual se deseja
trabalhar, começando com registro 0x3B (ACCEL_XOUT_H)
199   Wire.endTransmission(false);            // termina transmissão mas continua com I2C
aberto (envia STOP e START)
200   Wire.requestFrom(MPU_ADDR, 14);         // configura para receber 14 bytes
começando do registro escolhido acima (0x3B)
201
202   AcX = Wire.read() << 8;                 // lê primeiro o byte mais significativo
203   AcX |= Wire.read();                     // depois lê o bit menos significativo
204   _rawData.AcX = AcX;
205
206   AcY = Wire.read() << 8;
207   AcY |= Wire.read();
208   _rawData.AcY = AcY;
209
210   AcZ = Wire.read() << 8;
211   AcZ |= Wire.read();
212   _rawData.AcZ = AcZ;
213
214   Tmp = Wire.read() << 8;
215   Tmp |= Wire.read();
216   _rawData.Tmp = Tmp;
217
218   GyX = Wire.read() << 8;
219   GyX |= Wire.read();
220   _rawData.GyX = GyX;
221
222   GyY = Wire.read() << 8;
223   GyY |= Wire.read();
224   _rawData.GyY = GyY;
225
226   GyZ = Wire.read() << 8;
227   GyZ |= Wire.read();
228   _rawData.GyZ = GyZ;
229
230   led_state = !led_state;
231   digitalWrite(LED_BUILTIN, led_state);         // pisca LED do NodeMCU a cada
leitura do sensor
232
233   return _rawData;
234 }
235
236 /*
**********************************************************************************
*/
237 /* Method's name:          CalibrateGyro
    */
238 /* Description:            Set gyro calibration values for baseline shift
   */
239 /*
    */
240 /* Entry parameters:       float X -> X axis calibration value
    */
```

```
241 /*                      float Y -> Y axis calibration value
    */
242 /*                      float Z -> Z axis calibration value
    */
243 /*
    */
244 /* Return parameters:      n/a
    */
245 /*
*************************************************************************
*/
246 void IMU::CalibrateGyro(float X, float Y, float Z){
247   calGyX = X;
248   calGyY = Y;
249   calGyZ = Z;
250 }
251
252 /*
*************************************************************************
*/
253 /* Method's name:          CalibrateAcl
   */
254 /* Description:            Set accelerometer calibration values for baseline shift
    */
255 /*
    */
256 /* Entry parameters:       float X -> X axis calibration value
    */
257 /*                         float Y -> Y axis calibration value
    */
258 /*
    */
259 /* Return parameters:      n/a
    */
260 /*
*************************************************************************
*/
261 void IMU::CalibrateAcl(float X, float Y){
262   calAcX = X;
263   calAcY = Y;
264 }
265
266
267 /*
*************************************************************************
*/
268 /* Method's name:          getRawData
   */
269 /* Description:            Returns internal raw sensor data struct
    */
270 /*
    */
271 /* Entry parameters:
    */
272 /*
    */
273 /* Return parameters:      mpu -> raw data struct
   */
```

```
/*
*****************************************************************************
*/
mpu IMU::getRawData(){
  return _rawData;
}


/*
*****************************************************************************
*/
/* Method's name:          processMPUData
  */
/* Description:            Converts raw data to actual values. Also finds angular
  */
/*                         displacement
  */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      processedMpu -> processed data struct
   */
/*
*****************************************************************************
*/
processedMpu IMU::processMPUData(){

  /*Convertendo o dado lido do MPU-6050 de acordo com a
  sensibilidade do giroscopio*/
  _processedData.GyX = (float)(_rawData.GyX - calGyX)/131;
  _processedData.GyY = (float)(_rawData.GyY - calGyY)/131;
  _processedData.GyZ = (float)(_rawData.GyZ - calGyZ)/131;

  /*Convertendo o dado lido do MPU-6050 de acordo com a
  sensibilidade do acelerometro*/
  _processedData.AcX = (float)(_rawData.AcX - calAcX)/16384;
  _processedData.AcY = (float)(_rawData.AcY- calAcY)/16384;
  _processedData.AcZ = (float)(_rawData.AcZ)/16384;

  /*Utilizando as informacoes de velocidade angular e aceleracao
  para calcular a posicao angular*/
  processAngles(_processedData);

  /*Dado de temperatura da MPU-6050*/
  _processedData.Tmp = (float)_rawData.Tmp;



  return _processedData;
}


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

```
/* Method's name:          processAngles
   */
/* Description:            Converts gyro and accel data into angular displacement
  */
/*
   */
/* Entry parameters:       processedMpu dados -> data struct to process
  */
/*
   */
/* Return parameters:      n/a
   */
/*
*************************************************************************************
*/
void IMU::processAngles(processedMpu dados){

  unsigned long aux = micros();
  float x2, y2, z2, result;

  /*Calculo do tempo percorrido desde a ultima medida*/
  float dt = (float)(aux - _lastTimestamp)/1000000;

  /*Guardando o tempo desta medida*/
  _lastTimestamp = aux;

  //Serial.println(dt);

  /*Posicao angular medida somente pela integracao da velocidade*/
  _gyroPitch += _processedData.GyX*dt;  //Angulo de Pitch
  _gyroRoll  += _processedData.GyY*dt;  //Angulo de Roll

  /*Valores da ultima iteracao dos angulos*/
  _ang.GyPitch = _procAng.Pitch;
  _ang.GyRoll  = _procAng.Roll;
  _ang.GyYaw   = _procAng.Yaw;

  /*Posicao angular atraves da integracao da velocidade
  considerando a ultima iteracao*/
  _ang.GyPitch  += _processedData.GyX*dt; //MPU eixo X

  /*Compensacao de giro em Yaw na posicao angular*/
  if(yaw_compensation)
    _ang.GyPitch  += _ang.GyRoll*sin(_ang.GyYaw*dt*DEG_2_RAD);

   /*Posicao angular atraves da integracao da velocidade
  considerando a ultima iteracao*/
  _ang.GyRoll   += _processedData.GyY*dt; //MPU eixo Y

  /*Compensacao de giro em Yaw na posicao angular*/
  if(yaw_compensation)
    _ang.GyRoll   -= _ang.GyPitch*sin(_ang.GyYaw*dt*DEG_2_RAD);

  _ang.GyYaw    += _processedData.GyZ*dt; //MPU eixo Z

/*
  Serial.printf("Acelerometer INPUT:\t %f\t%f\t%f\n",_processedData.AcX,
_processedData.AcY, _processedData.AcZ);
*/
```

```
367
368   /*CALCULO DA POSICAO ANGULAR ATRAVES DO ACELEROMETRO*/
369
370   /*Elevando os dados ao quadrado*/
371   x2 = _processedData.AcX*_processedData.AcX;
372   y2 = _processedData.AcY*_processedData.AcY;
373   z2 = _processedData.AcZ*_processedData.AcZ;
374
375   /*Decomposicao do vetor da gravidade*/
376   //Pitch
377   result = sqrt(x2+z2);
378   /*Arco-Tangente para encontra o angulo apos decomposicao*/
379   _ang.AclPitch = -1*atan2(-1*_processedData.AcY, result)*RAD_2_DEG;
380
381   //Roll
382   /*Arco-Tangente para encontra o angulo apos decomposicao*/
383   _ang.AclRoll  = -1*atan2(_processedData.AcX, _processedData.AcZ)*RAD_2_DEG;
384
385   /*Aplicacao do filtro complementar para obter o angulo final*/
386   filterMPUData();
387
388   if(debbuging_enabled)
389     Serial.printf("%f,%f,%f\n",_gyroRoll, _ang.AclRoll, _procAng.Roll);
390
391 /*
392   Serial.printf("ANGULOS GIROSCOPIO:\nROLL:%f\tPITCH:%f\t%YAW:%f\n",_ang.GyRoll,
_ang.GyPitch, _ang.GyYaw);
393   Serial.printf("ANGULOS ACELEROMETRO:\nROLL:%f\tPITCH:%f\n",_ang.AclRoll,
_ang.AclPitch);
394   Serial.printf("ANGULOS COMPLEMENTAR:\nROLL:%f\tPITCH:%f\t%YAW:%f\n\n",
_procAng.Roll, _procAng.Pitch, _procAng.Yaw);
395 */
396 }
397
398
399 /*
**********************************************************************************
*/
400 /* Method's name:          filterMPUData
    */
401 /* Description:            Complementary filter to keep angular displacement from
   */
402 /*                         drifting
   */
403 /*
    */
404 /* Entry parameters:       n/a
    */
405 /*
    */
406 /* Return parameters:      n/a
    */
407 /*
**********************************************************************************
*/
408 void IMU::filterMPUData( ){
409
410   /*Atribuindo 'pesos' do filtro para as medidas de cada sensor
411   e construindo o sinal final.*/
```

```
412   _procAng.Roll   =   CF_GY*_ang.GyRoll + CF_AC*_ang.AclRoll;
413   _procAng.Pitch  =   CF_GY*_ang.GyPitch + CF_AC*_ang.AclPitch;
414   _procAng.Yaw    =   _ang.GyYaw;
415
416
417 }
418
419
420
421 /*
    ************************************************************************************
    */
422 /* Method's name:          update
      */
423 /* Description:            Reads from MPU-6050 and process data, updating internal
       */
424 /*                         values
      */
425 /*
       */
426 /* Entry parameters:       n/a
       */
427 /*
       */
428 /* Return parameters:      n/a
       */
429 /*
    ************************************************************************************
    */
430 void IMU::update(){
431
432   unsigned char aux = 0;
433   float auxRollVel = 0;
434   float auxPitchVel = 0;
435
436   readRawMPU();
437   processMPUData();
438
439
440   /*if(_meanPos == 9){
441     _meanPos = 0;
442   }
443
444   _roll_vel[_meanPos] = _processedData.GyY;
445   _pitch_vel[_meanPos] = _processedData.GyX;
446
447   while(aux != 10){
448     auxRollVel  += _roll_vel[aux];
449     auxPitchVel += _pitch_vel[aux];
450     aux++;
451   }
452
453   _meanVel.Roll = auxRollVel/10;
454   _meanVel.Pitch = auxPitchVel/10;
455
456   _meanPos++;*/
457
458   _gyroRollInput = (_gyroRollInput*0.8) + (_processedData.GyY*0.2);
459   _gyroPitchInput = (_gyroPitchInput*0.8) + (_processedData.GyX*0.2);
```

```
460   _gyroYawInput = (_gyroYawInput*0.8) + (_processedData.GyZ*0.2);
461
462 }
463
464 /*
    ********************************************************************************
    */
465 /* Method's name:          getPitchVel
       */
466 /* Description:            Returns gyro pitch velocity after complementary filter
      */
467 /*
       */
468 /* Entry parameters:       n/a
       */
469 /*
       */
470 /* Return parameters:      float -> Pitch velocity
       */
471 /*
    ********************************************************************************
    */
472 float IMU::getPitchVel(){
473   return _gyroPitchInput;
474
475 }
476
477 /*
    ********************************************************************************
    */
478 /* Method's name:          getRollVel
      */
479 /* Description:            Returns gyro roll velocity after complementary filter
       */
480 /*
       */
481 /* Entry parameters:       n/a
       */
482 /*
       */
483 /* Return parameters:      float -> Roll velocity
      */
484 /*
    ********************************************************************************
    */
485 float IMU::getRollVel(){
486   return _gyroRollInput;
487
488 }
489
490 /*
    ********************************************************************************
    */
491 /* Method's name:          getYawVel
       */
492 /* Description:            Returns gyro yaw velocity after complementary filter
      */
493 /*
       */
```

```
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      float -> Yaw velocity
   */
/*
*************************************************************************
*/
float IMU::getYawVel(){
  return _gyroYawInput;

}



/*
*************************************************************************
*/
/* Method's name:          getData
   */
/* Description:            Returns internal processed data struct
  */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      processedMpu -> internal processed data struct
  */
/*
*************************************************************************
*/
processedMpu IMU::getData(){

  return _processedData;

}


/*
*************************************************************************
*/
/* Method's name:          getRawAngles
  */
/* Description:            Returns raw angles from gyro and accelerometer calculation
  */
/*
   */
/* Entry parameters:       n/a
   */
/*
   */
/* Return parameters:      angles -> internal processed data struct
  */
/*
*************************************************************************
*/
```

```
528 angles IMU::getRawAngles(){
529   return _ang;
530 }
531
532
533 /*
    ********************************************************************************
    */
534 /* Method's name:          getRotations
      */
535 /* Description:            Returns processed angular displacement after the filter
       */
536 /*                         on Roll, Pitch and Yaw
      */
537 /*
       */
538 /* Entry parameters:       n/a
       */
539 /*
       */
540 /* Return parameters:      _procAng -> internal processed angular displacement struct
      */
541 /*
    ********************************************************************************
    */
542 processedAngles IMU::getRotations(){
543   return _procAng;
544 }
545
546 /*
    ********************************************************************************
    */
547 /* Method's name:          getGyroVel
      */
548 /* Description:            Returns the velocity mean from the gyroscope sensor
       */
549 /*                         on Roll, Pitch and Yaw
      */
550 /*
       */
551 /* Entry parameters:       n/a
       */
552 /*
       */
553 /* Return parameters:      gyroVel -> internal mean velocities struct
      */
554 /*
    ********************************************************************************
    */
555 gyroVel IMU::getGyroVel(){
556   return _meanVel;
557 }
558
559
560 /*
    ********************************************************************************
    */
561 /* Method's name:          enableDebug
       */
```

```
562 /* Description:            Enables serial communication for debbugging
       */
563 /*
       */
564 /* Entry parameters:       n/a
       */
565 /*
       */
566 /* Return parameters:      n/a
       */
567 /*
    ************************************************************************
    */
568 void IMU::enableDebug(){
569   debbuging_enabled = 1;
570 }
571
572
573 /*
    ************************************************************************
    */
574 /* Method's name:          disableDebug
      */
575 /* Description:            Disables serial communication for debbugging
      */
576 /*
       */
577 /* Entry parameters:       n/a
       */
578 /*
       */
579 /* Return parameters:      n/a
       */
580 /*
    ************************************************************************
    */
581 void IMU::disableDebug(){
582   debbuging_enabled = 0;
583 }
584
585 /*
    ************************************************************************
    */
586 /* Method's name:          disableYawComp
      */
587 /* Description:            Disables roll and pitch angle compensation using yaw
      */
588 /*
       */
589 /* Entry parameters:       n/a
       */
590 /*
       */
591 /* Return parameters:      n/a
       */
592 /*
    ************************************************************************
    */
593  void IMU::disableYawComp(){
```

```
  yaw_compensation = 0;
 }


/*
*****************************************************************************
*/
/* Method's name:           enableYawComp
   */
/* Description:             Enables roll and pitch angle compensation using yaw
   */
/*
   */
/* Entry parameters:        n/a
   */
/*
   */
/* Return parameters:       n/a
   */
/*
*****************************************************************************
*/
 void IMU::enableYawComp(){
   yaw_compensation = 1;
 }
```