main/Bateria.cpp

```cpp
/* ********************************************************************** */
/* File name:        Bateria.cpp                                          */
/* File description: This file contains the functions/methods for         */
/*                   initializing and using the battery                   */
/* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,          */
/*                   Guilherme Abreu                                      */
/* Creation date:    11out2021                                            */
/* Revision date:    28nov2021                                            */
/* ********************************************************************** */
#include "Bateria.h"

/* ********************************************************************** */
/* Method name:        Bateria                                            */
/* Method description: Declaring a Bateria class and initializing it      */
/* Input params: pino                                                     */
/* Output params: n/a                                                     */
/* ********************************************************************** */
Bateria:: Bateria(byte pino) {
  this->pino = pino;

//  void initLeitorBateria();
}

/* ********************************************************************** */
/* Method name:        initLeitorBateria                                  */
/* Method description: initializing battery sensor                        */
/* Input params: n/a                                                      */
/* Output params: n/a                                                     */
/* ********************************************************************** */
void Bateria:: initLeitorBateria(){
  pinMode(pino, INPUT); /*settng desired pin to be our level reader*/
}

/* ********************************************************************** */
/* Method name:        getNivelBateria                                    */
/* Method description: Get the current battery level                      */
/* Input params: n/a                                                      */
/* Output params: n/a                                                     */
/* ********************************************************************** */
int Bateria:: getNivelBateria() {
  int sensorBateria = analogRead(pino); /*Setting the analog read as the variable sensorBateria*/
  float fnivelBateria = (sensorBateria*100/1023.0); /*Convert the value to a percentage value (max value from analog pin is 1023)*
  int inivelBateria = (int) fnivelBateria; /*Convert variable type, from float to int*/

  return inivelBateria;
}

/* ********************************************************************** */
/* Method name:        checkAlertaBateria                                 */
/* Method description: Check battery level alert                          */
/* Input params: n/a                                                      */
/* Output params: true or false                                           */
/* ********************************************************************** */
boolean Bateria:: checkAlertaBateria() {
  if (getNivelBateria() <= 20){ /*using function getNivelBateria to get our battery level, and if its under 20% will give us an al
  return true;
  }else{
  return false;
  }
}
```

---

main/DriverBuzzer.cpp

```cpp
/* ********************************************************************** */
/* File name:        DriverBuuzer.ccp                                     */
/* File description: This file contains the functions/methods for         */
/*                   initializing and using the Buzzer                    */
/* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,          */
/*                   Guilherme Abreu                                      */
/* Creation date:    11out2021                                            */
/* Revision date:    28nov2021                                            */
/* ********************************************************************** */
#include "DriverBuzzer.h"
```

```
11
12  /* ********************************************************************* */
13  /* Method name:        DriverBuzzer                                      */
14  /* Method description: Declaring a DriverBuzzer class and                */
15  /*                     initializing it                                   */
16  /* Input params: pino                                                    */
17  /* Output params: n/a                                                    */
18  /* ********************************************************************* */
19  DriverBuzzer:: DriverBuzzer(byte pino) {
20    this->pino = pino;
21
22  //  void initBuzzer();
23  }
24
25  /* ********************************************************************* */
26  /* Method name:        initBuzzer                                        */
27  /* Method description: Initializing the Buzzer and setting is pin        */
28  /* Input params: n/a                                                     */
29  /* Output params: n/a                                                    */
30  /* ********************************************************************* */
31  void DriverBuzzer:: initBuzzer(){
32    pinMode(pino, INPUT_PULLUP); /*Defining select pin as a PULLUP Input */
33
34  }
35
36  /* ********************************************************************* */
37  /* Method name:        tocarAlarme                                       */
38  /* Method description: Start playing our Buzzer                          */
39  /* Input params: n/a                                                     */
40  /* Output params: n/a                                                    */
41  /* ********************************************************************* */
42  void DriverBuzzer:: tocarAlarme(){
43    tone(pino, 1000); /*Setting our alarm tone */
44    delay(1000); /*Delaying the alarm for 1000ms */
45    tone(pino,3000); /*Setting a different tone from before */
46    delay(1000); /*Delaying the alarm for 1000ms */
47  }
48
49  /* ********************************************************************* */
50  /* Method name:        pararAlarme                                       */
51  /* Method description: Stop playing our Buzzer                           */
52  /* Input params: n/a                                                     */
53  /* Output params: n/a                                                    */
54  /* ********************************************************************* */
55  void DriverBuzzer:: pararAlarme(){
56    noTone(pino); /*Setting no tone to Buzzer so it won't play anymore */
57  }
```

---

main/DriverEscovas.cpp

```
1   /* ********************************************************************* */
2   /* File name:        DriverEscovas.ccp                                   */
3   /* File description: This file contains the functions/methods for        */
4   /*                   initializing and using the driver for brushes       */
5   /* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,         */
6   /*                   Guilherme Abreu                                     */
7   /* Creation date:    11out2021                                           */
8   /* Revision date:    28nov2021                                           */
9   /* ********************************************************************* */
10  #include "DriverEscovas.h"
11
12  /* ********************************************************************* */
13  /* Method name:        DriverEscovas                                     */
14  /* Method description: Declaring a DriverEscovas class and               */
15  /*                     initializing it                                   */
16  /* Input params: pino1, pino2, pino3, pino4                              */
17  /* Output params: n/a                                                    */
18  /* ********************************************************************* */
19  DriverEscovas:: DriverEscovas(byte pino1, byte pino2, byte pino3, byte pino4) {
20    this->pino1 = pino1;
21    this->pino2 = pino2;
22    this->pino3 = pino3;
23    this->pino4 = pino4;
24
25  //  void initEscovas();
26  }
```

```
27
28  /* ************************************************************** */
29  /* Method name:        initEscovas                                */
30  /* Method description: initiliazing brushes                       */
31  /* Input params: n/a                                              */
32  /* Output params: n/a                                             */
33  /* ************************************************************** */
34  void DriverEscovas:: initEscovas(){
35    pinMode(pino1, OUTPUT); /*Setting pino1 as an Output */
36    pinMode(pino2, OUTPUT); /*Setting pino2 as an Output */
37    pinMode(pino3, OUTPUT); /*Setting pino3 as an Output */
38    pinMode(pino4, OUTPUT); /*Setting pino4 as an Output */
39
40    digitalWrite(pino1, LOW); /*Setting pino1 as LOW signal */
41    digitalWrite(pino2, LOW); /*Setting pino2 as LOW signal */
42    digitalWrite(pino3, LOW); /*Setting pino3 as LOW signal */
43    digitalWrite(pino4, LOW); /*Setting pino4 as LOW signal */
44
45  }
46
47  /* ************************************************************** */
48  /* Method name:        ligaMotores                                */
49  /* Method description: Turn on motors                             */
50  /* Input params: n/a                                              */
51  /* Output params: n/a                                             */
52  /* ************************************************************** */
53  void DriverEscovas:: ligaMotores(){
54    analogWrite(pino1, 0); /*Setting pino 1 as 0 on a analog level */
55    analogWrite(pino2, 210); /*Setting pino 2 as 210 on a analog level */
56    analogWrite(pino3, 210); /*Setting pino 3 as 210 on a analog level */
57    analogWrite(pino4, 0); /*Setting pino 4 as 0 on a analog level */
58  }
59
60  /* ************************************************************** */
61  /* Method name:        desligaMotores                             */
62  /* Method description: Turn off motors                            */
63  /* Input params: n/a                                              */
64  /* Output params: n/a                                             */
65  /* ************************************************************** */
66  void DriverEscovas:: desligaMotores(){
67    digitalWrite(pino1, LOW); /*Setting pino1 as LOW signal */
68    digitalWrite(pino2, LOW); /*Setting pino2 as LOW signal */
69    digitalWrite(pino3, LOW); /*Setting pino3 as LOW signal */
70    digitalWrite(pino4, LOW); /*Setting pino4 as LOW signal */
71  }
```

---

main/DriverHG7881.cpp

```
1  #include "DriverHG7881.h"
2
3  DriverHG7881:: DriverHG7881(byte pino) {
4    this->pino = pino;
5  }
```

---

main/DriverLcd.cpp

```
1
2
3  /* ************************************************************** */
4  /* File name:        DriverLcd.cpp                                */
5  /* File description: This file contains the functions/methods for */
6  /*                   initializing and using the battery           */
7  /* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,   */
8  /*                   Guilherme Abreu                              */
9  /* Creation date:    11out2021                                    */
10 /* Revision date:    28nov2021                                    */
11 /* ************************************************************** */
12 #include "DriverLcd.h"
13 #include <Arduino.h>
14 #include <Wire.h>
15 #include <LiquidCrystal_I2C.h>
16
17 /* ************************************************************** */
18 /* Method name:        initLcd                                    */
```

```
/* Method description: Initializing the LCD screen and printing      */
/*                     welcome message                                */
/* Input params: n/a                                                  */
/* Output params: n/a                                                 */
/* ****************************************************************** */



 DriverLcd:: DriverLcd(){
//  void initLcd();
this->lcd = LiquidCrystal_I2C(0x27,16,2);
}

void DriverLcd:: initLcd(){
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0,0);
  lcd.print("Welcome,");
  lcd.setCursor(0,1);
  lcd.print("We are Aspirabot!");
  delay(2000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Modo:");

}

/* ****************************************************************** */
/* Method name:        escreveModo                                    */
/* Method description: Write the mode our robot is working            */
/* Input params: modoOperacao                                         */
/* Output params: n/a                                                 */
/* ****************************************************************** */
void DriverLcd:: escreveModo( int modoOperacao){
  if(modoOperacao == HIGH){
        lcd.setCursor(0, 1);
        lcd.print("Seguidor Parede");
        delay(100);
      } else {
        lcd.setCursor(0, 1);
        lcd.print("Aleatorio      ");
        delay(100);
      }
}

/* ****************************************************************** */
/* Method name:        escreveAlerta                                  */
/* Method description: Write the alert when battery is about to die   */
/*                     or when stuck on loop                          */
/* Input params: n/a                                                  */
/* Output params: n/a                                                 */
/* ****************************************************************** */
void DriverLcd:: escreveAlerta(int tipoAlerta){
  lcd.setCursor(0, 1);
  switch(tipoAlerta){
    case 1:
        lcd.print("  ALERTA LOOP");
        delay(500);
        break;
        break;
    case 2:
        lcd.print(" ALERTA BATERIA");
        delay(500);
        break;
        break;
    default:
        lcd.print("    ALERTA");
        delay(500);
        break;
  }
}
```

main/DriverLocomocao.cpp

```
/* ****************************************************************** */
/* File name:        DriverLocomocao.cpp                              */
```

```
/* File description: This file contains the functions/methods for    */
/*                   initializing and using the locomotion           */
/* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,     */
/*                   Guilherme Abreu                                 */
/* Creation date:    11out2021                                       */
/* Revision date:    28nov2021                                       */
/* ********************************************************************* */
#include "DriverLocomocao.h"

/* ********************************************************************* */
/* Method name:        DriverLocomocao                                   */
/* Method description: Declaring a DriverLocomocao class and         */
/*                     initializing it                                 */
/* Input params: pino1, pino2, pino3, pino4                            */
/* Output params: n/a                                                  */
/* ********************************************************************* */
DriverLocomocao:: DriverLocomocao(byte pino1, byte pino2, byte pino3, byte pino4) {
  this->pino1 = pino1;
  this->pino2 = pino2;
  this->pino3 = pino3;
  this->pino4 = pino4;

//  void initLocomocao();
}

/* ********************************************************************* */
/* Method name:        initLocomocao                                     */
/* Method description: Initializing Locomotion                         */
/* Input params: n/a                                                   */
/* Output params: n/a                                                  */
/* ********************************************************************* */
void DriverLocomocao:: initLocomocao(){
  pinMode(pino1, OUTPUT); /* setting pino1 as Output */
  pinMode(pino2, OUTPUT); /* setting pino2 as Output */
  pinMode(pino3, OUTPUT); /* setting pino3 as Output */
  pinMode(pino4, OUTPUT); /* setting pino4 as Output */

  digitalWrite(pino1, LOW); /* setting pino1 as LOW signal */
  digitalWrite(pino2, LOW); /* setting pino2 as LOW signal */
  digitalWrite(pino3, LOW); /* setting pino3 as LOW signal */
  digitalWrite(pino4, LOW); /* setting pino4 as LOW signal */
}

/* ********************************************************************* */
/* Method name:        moverFrente                                       */
/* Method description: Moving robot forward                            */
/* Input params: n/a                                                   */
/* Output params: n/a                                                  */
/* ********************************************************************* */
boolean DriverLocomocao:: moverFrente(){
  //toda velocidade
  Serial.println("Andar Frente");
  analogWrite(pino1, 0); /* setting pino1 as 240 on an analog level */
  analogWrite(pino2, 240); /* setting pino2 as 0 on an analog level */
  analogWrite(pino3, 0); /* setting pino3 as 210 on an analog level */
  analogWrite(pino4, 210); /* setting pino4 as 0 on an analog level */
  //falta a parte do encoder
  return false;
}

/* ********************************************************************* */
/* Method name:        moverTras                                         */
/* Method description: Moving robot backwards                          */
/* Input params: n/a                                                   */
/* Output params: n/a                                                  */
/* ********************************************************************* */
boolean DriverLocomocao:: moverTras(){
  Serial.println("Andar Trás");
  analogWrite(pino1, 240); /* setting pino1 as 0 on an analog level */
  analogWrite(pino2, 0); /* setting pino2 as 240 on an analog level */
  analogWrite(pino3, 210); /* setting pino3 as 0 on an analog level */
  analogWrite(pino4, 0); /* setting pino4 as 210 on an analog level */

  return false;
}

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

```
/* Method name:        parar                                        */
/* Method description: stop the robot from moving                   */
/* Input params: n/a                                                */
/* Output params: n/a                                               */
/* ***************************************************************** */
void DriverLocomocao:: parar(){
  Serial.println("Parar");
  digitalWrite(pino1, LOW); /* setting pino1 as LOW signal */
  digitalWrite(pino2, LOW); /* setting pino2 as LOW signal */
  digitalWrite(pino3, LOW); /* setting pino3 as LOW signal */
  digitalWrite(pino4, LOW); /* setting pino4 as LOW signal */

}

/* ***************************************************************** */
/* Method name:        virarDireita                                 */
/* Method description: Turn robot to the right                      */
/* Input params: n/a                                                */
/* Output params: true or false                                     */
/* ***************************************************************** */
void DriverLocomocao:: virarDireita(){
  Serial.println("Virar Direita");
  analogWrite(pino1, 240); /* setting pino 1 as 0 on an analog level */
  analogWrite(pino2, 0); /* setting pino 2 as 0 on an analog level */
  analogWrite(pino3, 0); /* setting pino 3 as 210 on an analog level */
  analogWrite(pino4, 0); /* setting pino 4 as 0 on an analog level */

}

/* ***************************************************************** */
/* Method name:        virarEsquerda                                */
/* Method description: Turn robot to the left                       */
/* Input params: n/a                                                */
/* Output params:  true or false                                    */
/* ***************************************************************** */
void DriverLocomocao:: virarEsquerda(){
  Serial.println("Virar Esquerda");
  analogWrite(pino1, 0); /* setting pino 1 as 240 on an analog level */
  analogWrite(pino2, 0); /* setting pino 2 as 0 on an analog level */
  analogWrite(pino3, 0); /* setting pino 3 as 0 on an analog level */
  analogWrite(pino4, 210); /* setting pino 4 as 0 on an analog level */

}
```

main/Encoder.cpp

```
/* ***************************************************************** */
/* File name:        Encoder.ccp                                    */
/* File description: This file contains the functions/methods for   */
/*                   initializing and using the encoder             */
/* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,    */
/*                   Guilherme Abreu                                */
/* Creation date:    11out2021                                      */
/* Revision date:    28nov2021                                      */
/* ***************************************************************** */
#include "Encoder.h"

/* ***************************************************************** */
/* Method name:        Encoder                                      */
/* Method description: Declaring a Encoder class and initializing it */
/* Input params: pino1                                              */
/* Output params: n/a                                               */
/* ***************************************************************** */
Encoder:: Encoder(byte pino1) {
  this->pino1 = pino1;

//  void initEncoder();
}

/* ***************************************************************** */
/* Method name:        initEncoder                                  */
/* Method description: Iniializing the encoder                      */
/* Input params: n/a                                                */
/* Output params: n/a                                               */
/* ***************************************************************** */
void Encoder:: initEncoder(){
```

```
  pinMode(pino1, INPUT_PULLUP); /* defining selected pin as a PULLUP input */
}

/* ************************************************************** */
/* Method name:        getVeocity                                 */
/* Method description: Get current robot velocity                 */
/* Input params: n/a                                              */
/* Output params: n/a                                             */
/* ************************************************************** */
float Encoder:: getVelocity(){
  int pulse = digitalRead(pino1); /* get the velocity value from the selected pin */
  return pulse;
}

/* ************************************************************** */
/* Method name:        isMoving                                   */
/* Method description: return true if motor is moving             */
/* Input params: m/a                                              */
/* Output params: true or false                                   */
/* ************************************************************** */
boolean Encoder:: isMoving(){
  boolean isMoving = false;

  int counter = 0;
  int pulses = 0;

  int prevPulse = digitalRead(pino1); /* get pulse value from pino1 */
  int pulse = digitalRead(pino1); /* get another pulse value from pino1 */

  while (!isMoving){
    pulse = digitalRead(pino1); /* get pulse value from pino1 */
    delay(10); /* delay 10 miliseconds */
    if(prevPulse != pulse){ /* check previous pulse with actual pulse */
      delay(10);
      pulses++; /* if they are different add to pulses*/
    }else{
      counter++; /*if they are equal add to counter */
    }

    prevPulse = pulse;
    if(pulses > 3){ /* check if number of counted pulses is greater than 3 */
      delay(10);
      isMoving = true; /*if its true then our robot is moving */

    }
    if(counter > 200){ /* check if number of counter is geater than 200 */
      break; /*the robot is not moving */
    }
  }

  return isMoving;
}
```

main/SensorDistancia.cpp

```
/* ************************************************************** */
/* File name:        SensorDistancia.cpp                          */
/* File description: This file contains the functions/methods for */
/*                   initializing and using the sensor for distance */
/* Author name:      Andre Won, Cassio Dezzoti, Totmes Scheffer,  */
/*                   Guilherme Abreu                              */
/* Creation date:    11out2021                                    */
/* Revision date:    28nov2021                                    */
/* ************************************************************** */
#include "sensorDistancia.h"
#include<Arduino.h>

/* ************************************************************** */
/* Method name:        SensorDistancia                            */
/* Method description: Declaring a SensorDistancia class and      */
/*                     initializing it                            */
/* Input params: n/a                                              */
/* Output params: n/a                                             */
/* ************************************************************** */
SensorDistancia:: SensorDistancia(byte pino1, byte pino2) {
  this->pino1 = pino1;
```

```
  this->pino2 = pino2;

//  void initSensor();
}

/* ********************************************************************* */
/* Method name:        initSensor                                        */
/* Method description: Initializing the sensor                           */
/* Input params: n/a                                                     */
/* Output params: n/a                                                    */
/* ********************************************************************* */
void SensorDistancia:: initSensor(){
  pinMode(pino1,INPUT);   /* Echo */
  pinMode(pino2,OUTPUT);  /* Trigger */

  digitalWrite(pino1,LOW); /* defining pino1 as LOW signal */
  digitalWrite(pino2,LOW); /* defining pino2 as LOW signal */
}

/* ********************************************************************* */
/* Method name:        checkDistance                                     */
/* Method description: Compare the distance to the sensor with the   */
/*                     value from input                                  */
/* Input params: n/a                                                     */
/* Output params: true or false                                          */
/* ********************************************************************* */
boolean SensorDistancia:: checkDistance(float distancia){

    if (distancia < 5){
    return true;
    }else{
  return false;
    }

}

/* ********************************************************************* */
/* Method name:        getDistance                                       */
/* Method description: Get distance to the sensor                        */
/* Input params: m/a                                                     */
/* Output params: dist(float)                                            */
/* ********************************************************************* */
float SensorDistancia:: getDistance(){

  float tempo;

  digitalWrite(pino2,LOW); /* setting pino2 as LOW sginal */
  delayMicroseconds(5); /*delay 5 miliseconds */
  digitalWrite(pino2, HIGH); /* setting pino2 as HIGH signal */
  delayMicroseconds(10); /*delay 10 miliseconds */
  digitalWrite(pino2,LOW); /*setting pino2 as LOW signal */

  digitalWrite(pino1,LOW); /* setting pino1 as LOW signal */
  tempo = pulseIn(pino1,HIGH); /* get time that pino1 signal is HIGH (echo from ultrasound) */

  float dist;
  dist = 100*(tempo*0.00034029)/2; /*calculate distance using sound velocity in miliseconds */
                                   /* and multiply by 100 to get values in cm */

  return dist;

}
```