Apostila de Firmware

Miguel Abdala

2026

Inatel

Conteúdo

1 Instalações

1.1 Arduino IDE

A IDE (*Integrated Development Environment*) Arduino é um software que proporciona um ambiente de desenvolvimento completo para programar microcontroladores Arduino, ESP32, entre outros. É onde se escreve, edita, compila e carrega códigos (*sketches*) para o robô. Para fazer o download e a instalação do aplicativo, siga as instruções abaixo:

1. Acesse a página de download da IDE Arduino, clicando aqui;
2. Selecione o sistema operacional utilizado pelo seu computador. Para computadores Windows, selecione a primeira opção, conforme demonstrado na Figura 1 abaixo;

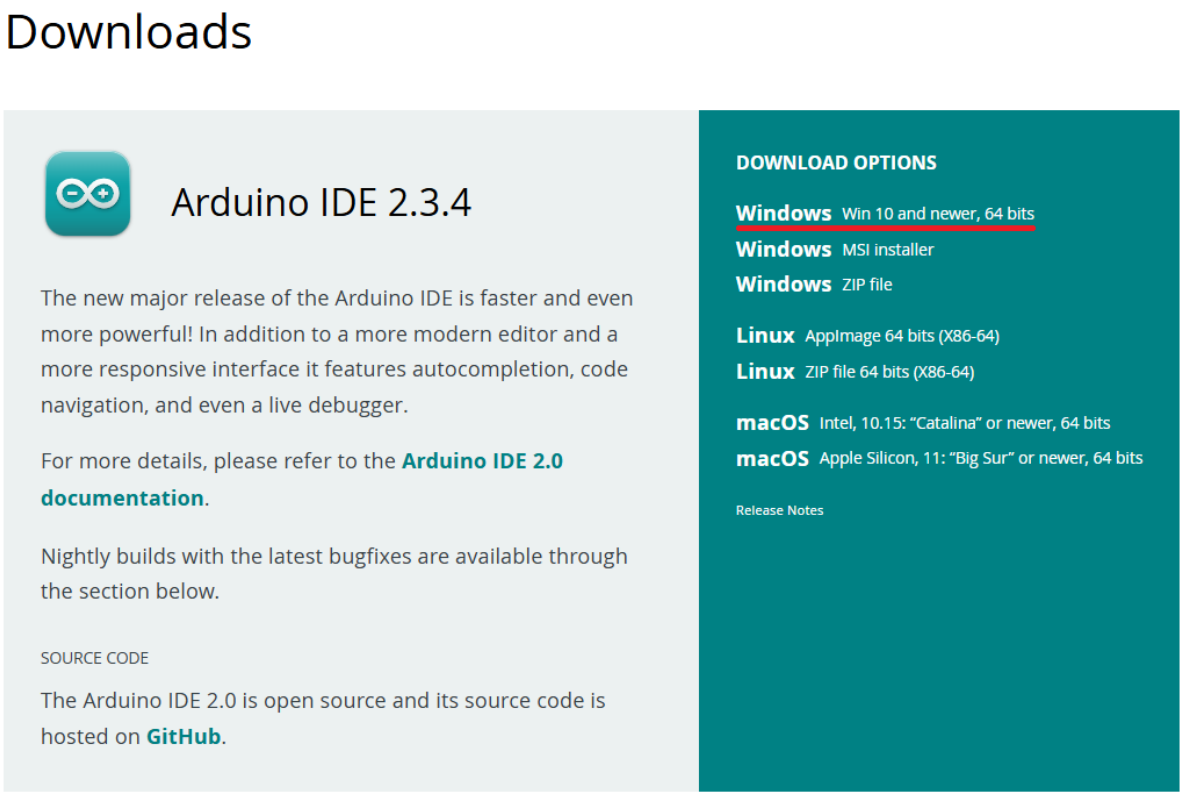

Figura 1: Página de seleção do sistema operacional

3. Na página para a qual você foi redirecionado, selecione a opção JUST DOWNLOAD. Você será redirecionado para outra página;

Figura 2: Página Just Download 1

4. Selecione novamente a opção JUST DOWNLOAD. Feito isso, o arquivo será baixado;

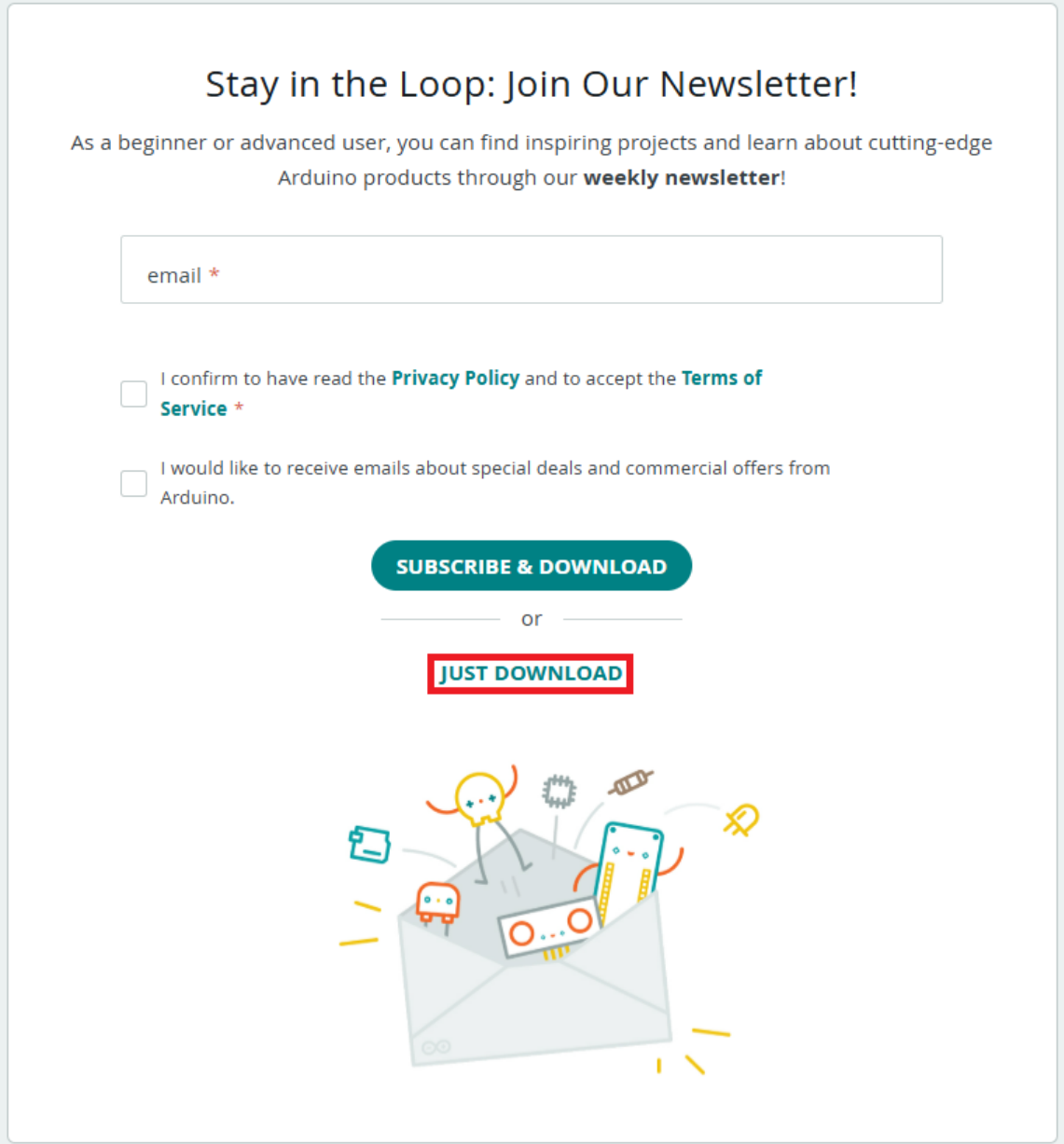

Figura 3: Página Just Download 2

5. Acesse a pasta onde o arquivo baixado foi salvo e dê um duplo clique nele para executá-lo;

6. Na janela que abrir, clique em Eu Concordo, para concordar com os Termos de Licença do programa;

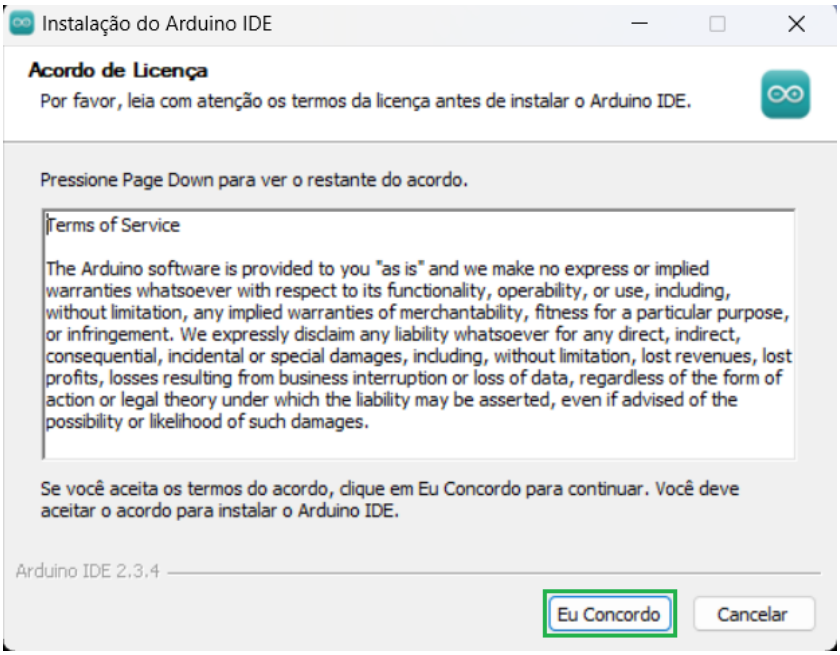

Figura 4: Concordância com os Termos de Licença

7. Selecione para quais usuários de seu computador o aplicativo deve ser instalado. Recomenda-se instalar apenas para seu usuário. Feita a seleção. Clique em Próximo;

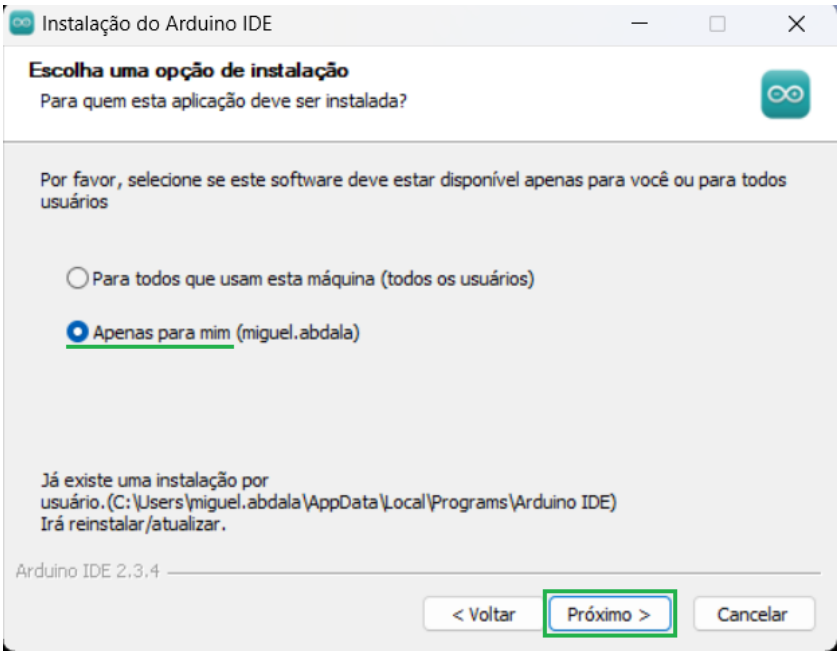

Figura 5: Opções de instalação

8. Escolha o local de instalação do programa. Recomenda-se deixar a Pasta de Destino padrão escolhida pelo próprio instalador. Clique em Instalar;

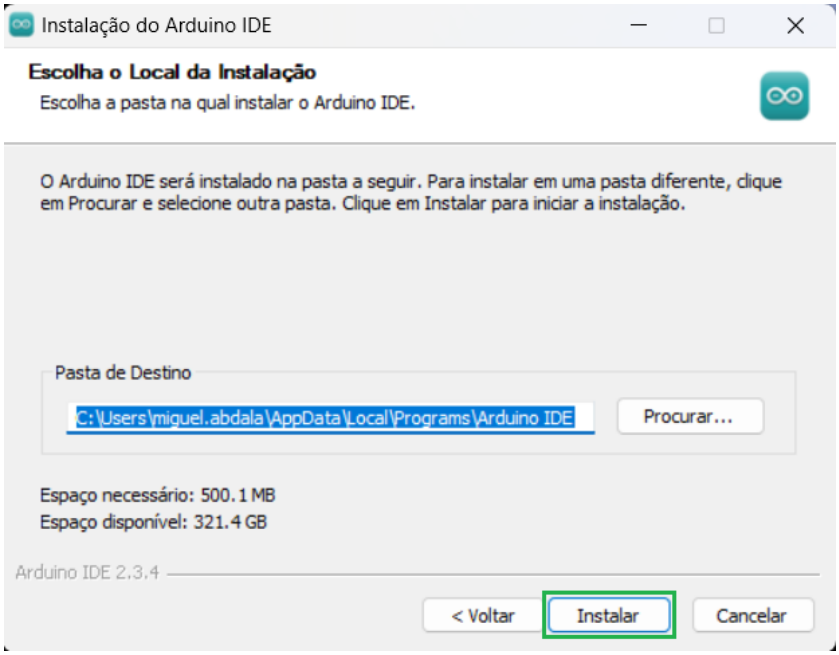

Figura 6: Seleção do Local de Instalação do programa

9. Finalizada a instalação, clique em Concluir. A IDE Arduino abrirá logo em sequência.

Figura 7: Concluindo instalação

1.2 Driver CP210x

O driver especifica para o computador a forma correta de se fazer a comunicação com a ESP32, permitindo o correto carregamento do código do robô no microcontrolador. Caso não seja instalado, o computador não irá reconhecer a ESP32 ao conectá-la via USB. O driver a ser instalado é o CP210x. As instruções abaixo demonstram como fazer a instalação:

1. Acesse a página de download, clicando **aqui**;
2. Selecione o sistema operacional utilizado pelo seu computador. Para computadores Windows, selecione a primeira opção, conforme demonstrado na Figura 8 abaixo. Ao clicar em uma das opções, você será redirecionado para outra página;

Figura 8: Download do driver ESP32

3. Abra a pasta onde o arquivo foi salvo. O arquivo baixado está no formato .zip. É necessário extraí-lo antes de prosseguir. Para tal, clique sobre ele com o botão direito do mouse e depois clique em extrair;
4. Abra a pasta originada a partir da extração do arquivo .zip, clique no arquivo de nome silabser com o botão direito e clique em instalar;

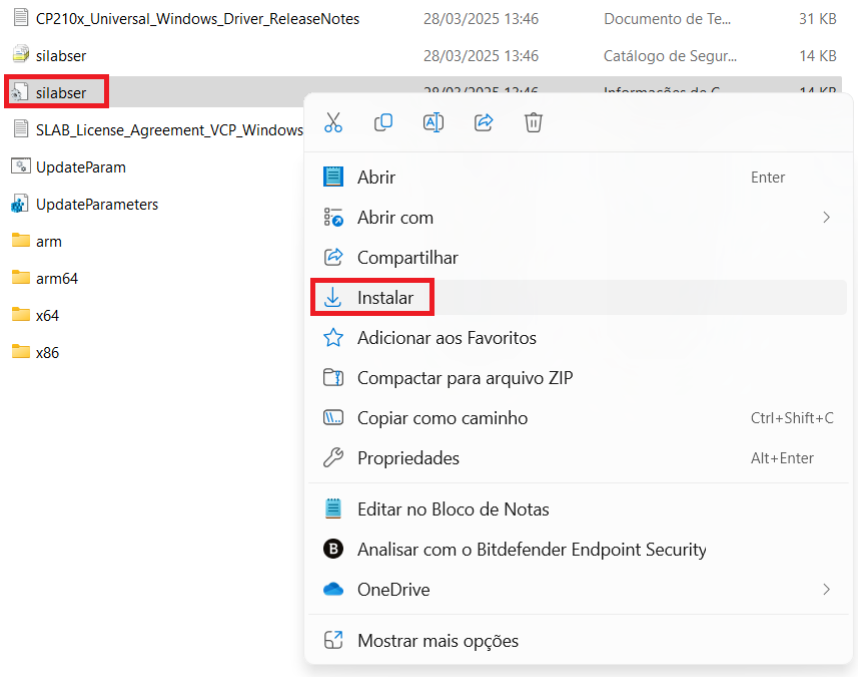

Figura 9: Iniciando instalação do driver

5. Na janela que aparecer, clique em Abrir. Caso uma janela de notificação do Windows abra logo em sequência, pressione Sim. Feito isso, o driver estará instalado;

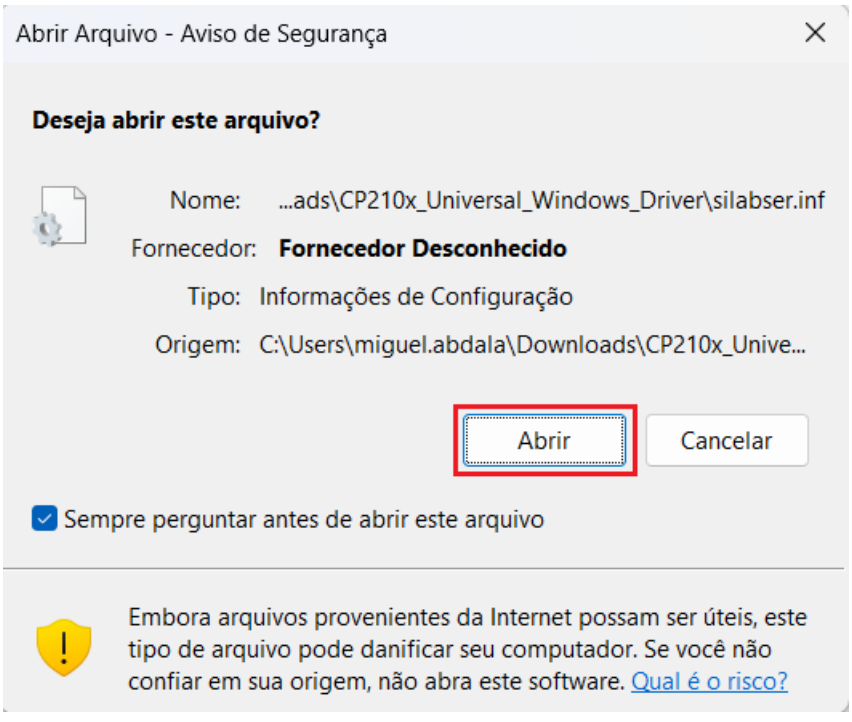

Figura 10: Instalando driver

6. Clique em Ok para finalizar o processo.

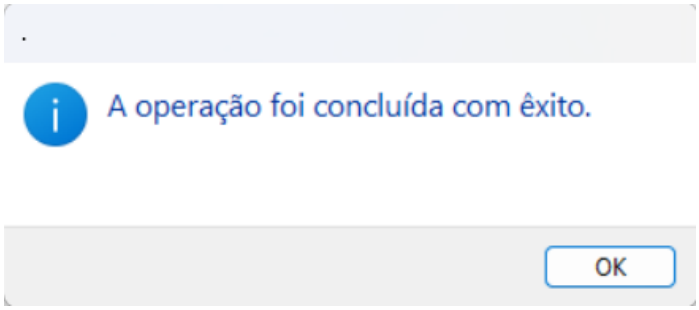

Figura 11: Driver instalado com sucesso

1.3 Bibioteca Bluepad32

Bibliotecas, no contexto de programação, são conjuntos de códigos prontos que facilitam a criação de projetos, evitando que o programador precise escrever tudo do zero. Por exemplo, se você quiser usar um sensor de temperatura, pode instalar uma biblioteca específica que já tem os comandos necessários para fazer o sensor funcionar, economizando tempo. Nesta aplicação, iremos utilizar a biblioteca Bluepad32 para fazer a conexão da ESP32 com o controle Bluetooth.

O passo a passo abaixo demonstra como fazer a instalação da biblioteca. Ele foi feito com base na documentação original - tópico *Arduino + ESP32 board - Option A* -, disponível no site da biblioteca (clique **aqui** para ir à página).

1. Abra a IDE Arduino. No canto superior esquerdo, clique em *File* (Arquivo) e selecione a opção *Preferences* (Preferências);

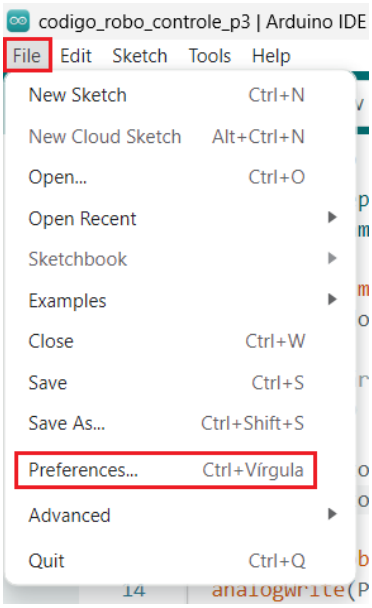

Figura 12: Abrindo campo de Preferências da IDE

2. Na janela que abrir, selecione a opção *Settings* (Configurações). Depois, clique no botão identificado na Figura 13 abaixo;

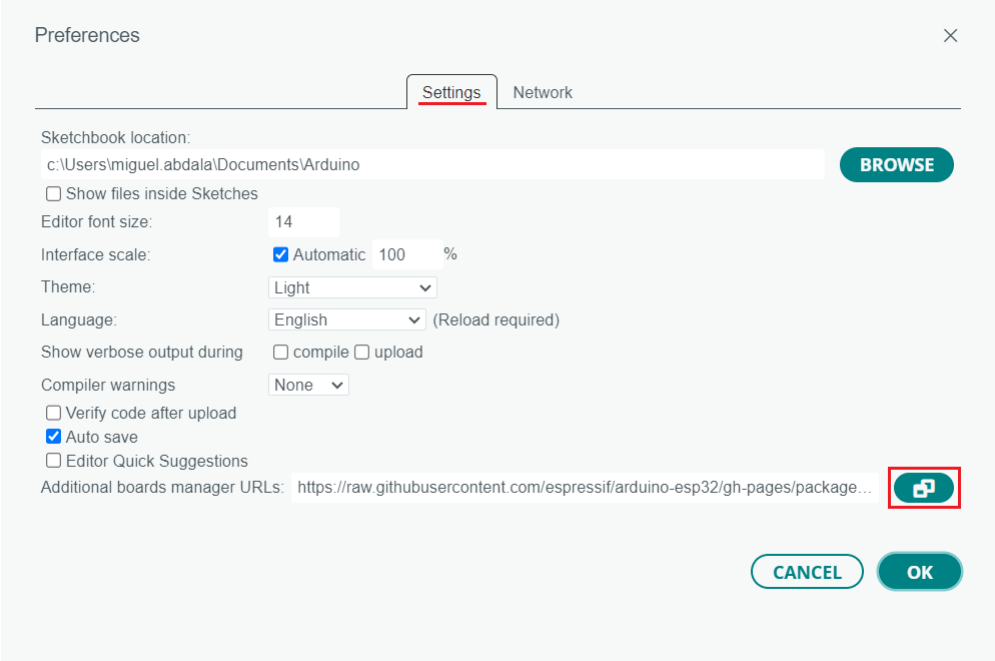

Figura 13: Janela de Preferências

3. Cole os links abaixo no campo que aparecer, um embaixo do outro. Esses links indicam à IDE Arduino onde procurar os arquivos da biblioteca no momento em que a instalação for solicitada;

   `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`

   `https://raw.githubusercontent.com/ricardoquesada/esp32-arduino-lib-builder/master/bluepad32_files/package_esp32_bluepad32_index.json`

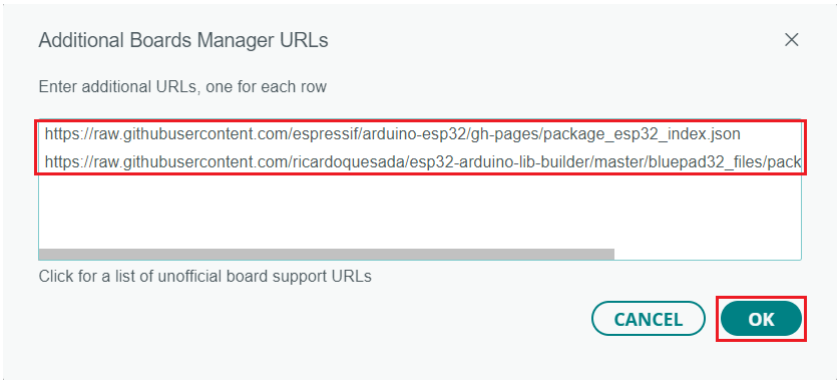

Figura 14: Identificação dos locais de download da biblioteca

4. Novamente na barra superior, selecione agora a opção *Tools* (Ferramentas). Clique na opção *Board* (Placa) e depois em *Board Manager* (Gerenciador de Placas);

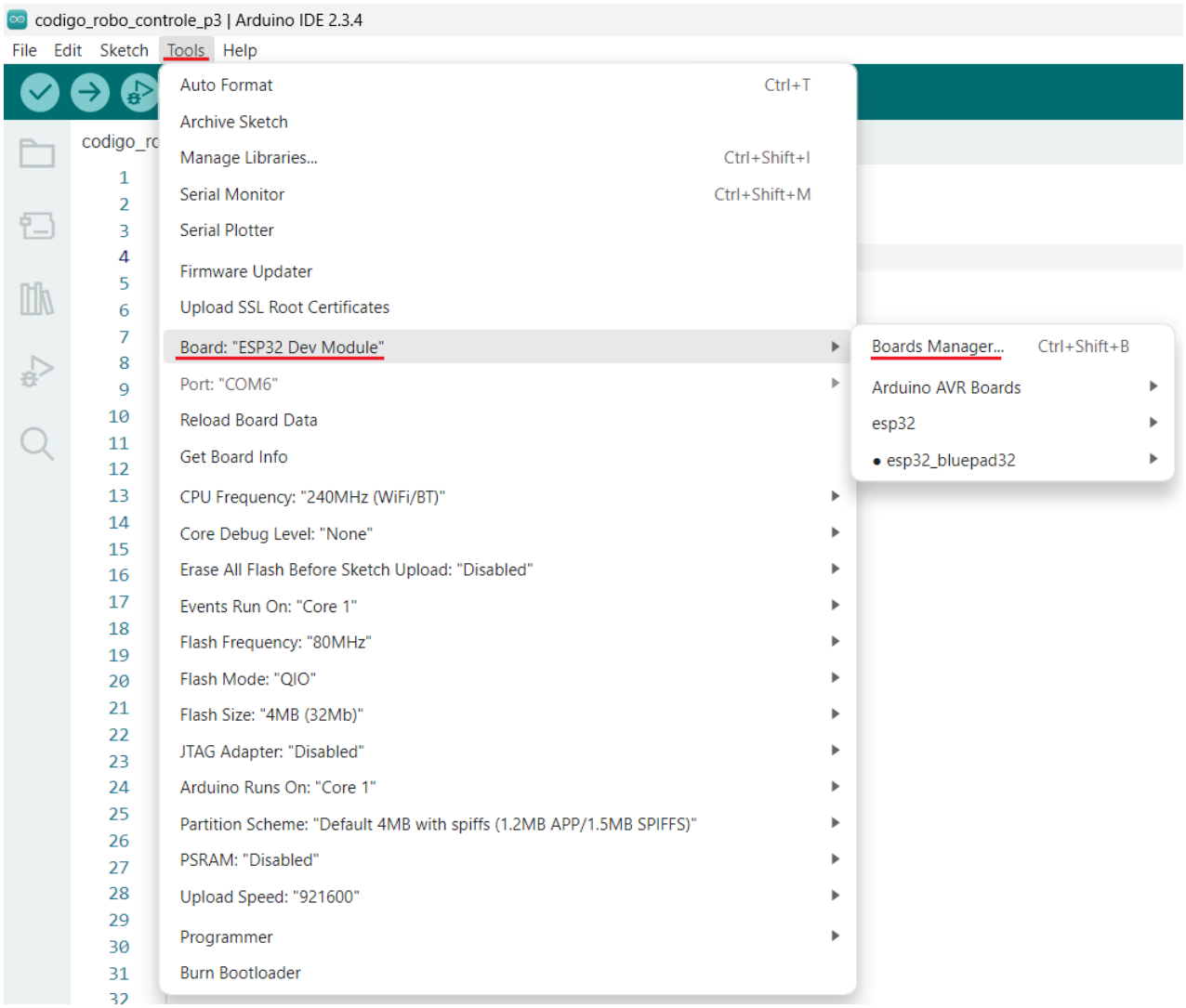

Figura 15: Abrindo o gerenciador de placas

5. Pesquise por ESP32 e instale as bibliotecas **esp32** e **esp32_bluepad32**. **Instale uma biblioteca de cada vez**, caso contrário, erros podem acontecer durante a instalação;

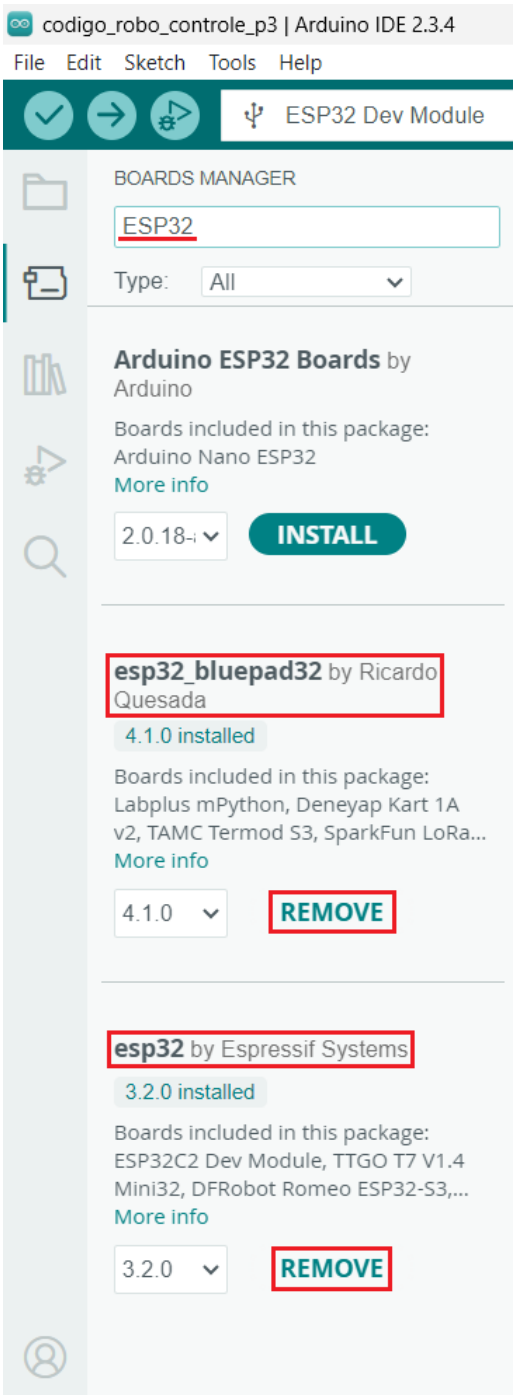

Figura 16: Instalando placas

Observações:

- A biblioteca esp32 é uma biblioteca base utilizada pela Bluepad32; por isso, deve-se instalá-la;
- As bibliotecas já estavam instaladas; por isso, aparece o *REMOVE* (Remover) na Figura 16 ao invés de *INSTALL* (Instalar).

2 Linguagem Arduino

A linguagem de programação utilizada para programar a ESP32 foi a linguagem nativa da plataforma Arduino. Nos links abaixo, é possível encontrar um tutorial completo explicando todas as funcionalidades da linguagem, juntamente com exemplos de implementação.

- Variáveis (clique **aqui** para ir à página)
- Estruturas (clique **aqui** para ir à página)
- Funções (clique **aqui** para ir à página)

Para entender melhor o código do robô, recomenda-se o seguinte roteiro de estudos:

1. Estruturas - Sketch
2. Variáveis - Tipos de Dados
3. Estruturas - Operadores Aritméticos
4. Estruturas - Outros Elementos da Sintaxe
5. Estruturas - Estruturas de Controle - IF/ELSE
6. Estruturas - Operadores de Comparação
7. Estruturas - Operadores Booleanos
8. Funções - Entradas e Saídas Digitais
9. Variáveis - Constantes
10. Funções - Entradas e Saídas Analógicas
11. Funções - Funções Matemáticas

3 Código principal do robô

3.1 Importando Bibliotecas

Nas duas primeiras linhas do programa, há a inclusão da biblioteca utilizada para fazer a comunicação com o controle de PS3 (Bluepad32) e a inclusão do arquivo de configurações. O arquivo de configurações será explicado posteriormente.

```
#include <Bluepad32.h>
#include "parametros.h"
```

Figura 17: Importando biblioteca e arquivo de configurações

3.2 Definição de variáveis

Após as inclusões, oito variáveis utilizadas ao longo do programa são declaradas. Abaixo consta uma explicação sobre cada variável.

- myControllers: representa de forma lógica/virtual o controle bluetooth utilizado para controlar o robô;
- roboLigado: indicador do estado do robô ligado;
  - ligado = verdadeiro / true
  - desligado = falso / false
- sentidoMotorEsquerdo e velocidadeMotorEsquerdo: indicam os pinos nos quais os terminais do motor esquerdo do robô estão conectados;
- sentidoMotorDireito e velocidadeMotorDireito: indicam os pinos nos quais os terminais do motor direito do robô estão conectados;
- direitoVesquerdoH e direitoHesquerdoV: indicam a forma de pilotagem do robô;
  - *joystick* direito funcionando na vertical e esquerdo, na horizontal
  - *joystick* direito funcionando na horizontal e esquerdo, na vertical

```
4   ControllerPtr myControllers[BP32_MAX_GAMEPADS];
5   bool roboLigado;
6
7   /* Foi necessário utilizar essas variáveis para permitir a
8      inversão do sentido de giro de cada motor de locomoção */
9
10  int sentidoMotorEsquerdo, velocidadeMotorEsquerdo;
11  int sentidoMotorDireito, velocidadeMotorDireito;
12
13  // Variáveis para inversão dos analógicos de movimentação
14  bool direitoVesquerdoH, direitoHesquerdoV;
```

Figura 18: Declaração de variáveis

3.3 Função desligaRobo()

Esta função contém todos os comandos necessários para interromper o funcionamento do robô. Nas primeiras linhas da função, utiliza-se o comando analogWrite (escrita analógica) para desligar (0) as saídas associadas ao controle dos motores de arma e de locomoção (todos os motores são configurados com velocidade 0). Ao final, define-se o estado do robô como desligado (roboLigado = false).

```
13  void desligaRobo() {
14    analogWrite(PINO_1_ARMA1, 0);
15    analogWrite(PINO_2_ARMA1, 0);
16
17    analogWrite(PINO_1_ARMA2, 0);
18    analogWrite(PINO_2_ARMA2, 0);
19
20    analogWrite(sentidoMotorDireito, 0);
21    analogWrite(velocidadeMotorDireito, 0);
22
23    analogWrite(sentidoMotorEsquerdo, 0);
24    analogWrite(velocidadeMotorEsquerdo, 0);
25
26    roboLigado = false;
27  }
```

Figura 19: Função desligaRobo()

3.4 Função onConnectedController()

Estabelece a ligação virtual entre o controle e a ESP32. A execução da função é feita de forma automática pela ESP32 sempre que um controle é conectado. Caso deseje realizar alguma tarefa assim que o controle for conectado, basta adicionar o trecho de código no local indicado (entre o comentário e o comando break).

```
29  void onConnectedController(ControllerPtr ctl) {
30
31    bool foundEmptySlot = false;
32
33    for (int i = 0; i < BP32_MAX_GAMEPADS; i++) {
34      if (myControllers[i] == nullptr) {
35        Serial.println("AVISO: controle conectado.");
36        myControllers[i] = ctl;
37        foundEmptySlot = true;
38
39        /* Caso deseje realizar alguma tarefa assim que a conexão
40           com o contole for estabelecidada, coloque o código aqui. */
41
42        break;
43      }
44    }
45
46    if (!foundEmptySlot) {
47      Serial.println("AVISO: Nao foi possivel conectar o controle.");
48      Serial.println("AVISO: Reinicie a ESP32 e tente novamente.");
49    }
50  }
```

Figura 20: Função onConnectedController()

3.5 Função onDisconnectedController()

Rompe a ligação virtual entre o controle e a ESP32. A execução da função é feita de forma automática pela ESP32 sempre que um controle é desconectado. Caso deseje realizar alguma tarefa assim que o controle for desconectado, basta adicionar o trecho de código no local indicado (entre o comentário e o comando break).

```
52  void onDisconnectedController(ControllerPtr ctl) {
53    for (int i = 0; i < BP32_MAX_GAMEPADS; i++) {
54      if (myControllers[i] == ctl) {
55        Serial.printf("AVISO: controle desconectado");
56        myControllers[i] = nullptr;
57        desligaRobo();
58
59        /* Caso deseje realizar alguma tarefa assim que o
60           controle for desconectado, coloque o código aqui */
61
62        break;
63      }
64    }
65  }
```

Figura 21: Função onDisconnectedController()

Tanto a função onConnectedController() quanto a função onDisconnectedController() são definidas pela própria Bluepad32. Entretanto, algumas modificações foram feitas a fim de deixá-las mais simples e compactas.

Uma chamada da função desligaRobo() foi adicionada à função de desconexão do controle para garantir que o robô pare sua movimentação e armas em caso de perda de conexão com o controle (sistema de proteção exigido nas regras de competição da categoria cupim - *fail safe*).

3.6 Função processControllers()

Responsável por mapear o que foi pressionado no controle e, a partir disso, determinar e controlar o funcionamento dos motores de arma e locomoção. **Toda a lógica de funcionamento do robô está descrita dentro desta função.**

3.6.1 Verificação de conexão

A primeira tarefa realizada pela função é confirmar se há um dispositivo bluetooth conectado, se este está enviando dados e se ele é de fato um controle (Gamepad). Essa última verificação é feita pois a mesma biblioteca pode ser utilizada para a conexão com diversos tipos de dispositivos bluetooth (mouse, teclado, ...). **Todos os trechos de código (processamento das informações do controle) após essa condição são executados apenas se ela for verdadeira**.

```
void processControllers() {
  for (auto myController : myControllers) {

    if (myController && myController->isConnected()
        && myController->hasData() && myController->isGamepad()) {

      /* A partir daqui inicia-se a lógica de funcionamento do robô.
         Qualquer alteração / nova implementação deve ser feita aqui. */
```

Figura 22: Verificação de conexão

3.6.2 Liga/Desliga robô

Feita a verificação da conexão, verifica-se o estado dos botões Select e Start. Se Select for pressionado (miscSelect() retorna *true* / verdadeiro), o robô é desligado (roboLigado = *false*). Já se Start for pressionado (miscStart() retorna *true* / verdadeiro), o robô é ligado (roboLigado = *true*).

O controle dos motores de locomoção e de arma só funciona se o robô estiver ligado, caso contrário, todos os motores são desligados pela função desliga robô.

```
// Se SELECT for presionado, desliga robô.
if (myController->miscSelect()) {
  roboLigado = false;
  Serial.println("Robo desligado.");
}

// Se START for presionado, liga robô.
else if (myController->miscStart()) {
  roboLigado = true;
  Serial.println("Robo ligado.");
}
```

Figura 23: Liga / Desliga robô

3.6.3 Arma

Os motores de arma são comandados pelos botões descritos abaixo.

- BOLINHA - b() == *true* - liga motores em um sentido
- QUADRADO - x() == *true* - liga motores em outro sentido
- TRIÂNGULO - y() == *true* - desliga motores

Recomenda-se **desligar os motores antes de inverter o sentido de rotação**. Ao inverter o sentido com os motores rotacionando, pode-se danificá-los mecanicamente.

```
/* ---------------------- Lógica de funcionamento da arma ---------------------- */

// Se BOLINHA for pressionado, roda arma para um lado.
if (myController->b()) {
  Serial.print("Arma Sentido 1\n");
  analogWrite(PINO_1_ARMA1, 0);
  analogWrite(PINO_2_ARMA1, MAX_PWM);
  analogWrite(PINO_1_ARMA2, MAX_PWM);
  analogWrite(PINO_2_ARMA2, 0);
}

// Se QUADRADO for pressionado, roda arma para o outro lado.
if (myController->x()) {
  Serial.print("Arma Sentido 2\n");
  analogWrite(PINO_1_ARMA1, MAX_PWM);
  analogWrite(PINO_2_ARMA1, 0);
  analogWrite(PINO_1_ARMA2, 0);
  analogWrite(PINO_2_ARMA2, MAX_PWM);
}

// Se TRIÂNGULO for presionado, desliga motores da arma.
if (myController->y()) {
  Serial.print("Arma desligada\n");
  analogWrite(PINO_1_ARMA1, 0);
  analogWrite(PINO_2_ARMA1, 0);
  analogWrite(PINO_1_ARMA2, 0);
  analogWrite(PINO_2_ARMA2, 0);
}
```

Figura 24: Controle dos motores de arma

A biblioteca segue como referência o controle de Xbox, por isso as funções são b(), x() e y(). Para identificar quais funções se referem aos botões do controle de PS3, basta compará-los.

3.6.4 Movimentação

Quando movimentados, os *joysticks* retornam um valor proporcional ao quão pressionados estão em cada direção (vertical ou horizontal). De acordo com a documentação da Bluepad32, para ambas as direções (horizontal e vertical), o valor retornado pode variar de -511 a 512. O sinal do valor depende do sentido do movimento. Entretanto, os valores podem variar em função do controle utilizado. Para o controle utilizado, tem-se o seguinte comportamento:

- esquerda: retorna um valor horizontal negativo
- direita: retorna um valor horizontal positivo
- cima: retorna um valor vertical negativo
- baixo: retorna um valor vertical positivo

No código do robô, os motores de movimentação podem ser controlados pelo movimento vertical do *joystick* direito e o movimento horizontal do *joystick* esquerdo, ou o contrário. Por padrão, é adotada a primeira configuração mencionada, sendo possível alterá-la por meio dos botões R1 e L1, conforme demonstrado na Figura 25 abaixo.

```
/* ----------------- Lógica de inversão dos analógicos de movimentação ----------------- */

// Se L1 for pressionado, locomoção Direito V - Esquerdo H
if (myController->l1()) {
  direitoVesquerdoH = true, direitoHesquerdoV = false;
}

// Se R1 for pressionado, locomoção Direito H - Esquerdo V
if (myController->r1()) {
  direitoVesquerdoH = false, direitoHesquerdoV = true;
}
```

Figura 25: Inversão dos *joysticks* de movimentação

Os valores vertical e horizontal do *joystick* esquerdo são retornados pelas funções axisY() e axisX(), respectivamente. Já para o *joystick* direito, têm-se as funções axisRY() e axisRX() (R = *right* = direito).

```
/* ----------------- Lógica de funcionamento da movimentação ----------------- */

int32_t valorAnalogicoV, valorAnalogicoH;

if (direitoVesquerdoH) {
  // Lê valor em Y do analógico direito (R-right).
  valorAnalogicoV = myController->axisRY();

  // Lê valor em X do analógico esquerdo (L-left).
  valorAnalogicoH = myController->axisX();
}

if (direitoHesquerdoV) {
  // Lê valor em X do analógico direito (R-right).
  valorAnalogicoH = myController->axisRX();

  // Lê valor em Y do analógico esquerdo (L-left).
  valorAnalogicoV = myController->axisY();
}
```

Figura 26: Leitura dos *joysticks*

Abaixo há uma explicação da lógica de movimentação projetada para o robô em função da movimentação dos *joysticks* do controle.

- **Trás : Direita** - Se os *joysticks* vertical e horizontal forem pressionados para trás (+) e para a direita (+), respectivamente, o robô deve andar para trás e para a direita. Neste caso, o sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda direita deve ser menor que a velocidade da roda esquerda.

```
// Analógico Y movimentado para trás
if (valorAnalogicoV > (PARADO_JOYSTICK_Y + TOLERANCIA_JOYSTICK)) {
  pwmMotorDireito1 = MIN_PWM;
  pwmMotorEsquerdo2 = MIN_PWM;

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {
    pwmMotorDireito2 = map(valorAnalogicoV - valorAnalogicoH,
                           PARADO_JOYSTICK_Y - MAX_JOYSTICK_X,
                           MAX_JOYSTICK_Y - PARADO_JOYSTICK_X,
                           MIN_PWM, MAX_PWM);
    pwmMotorEsquerdo1 = map(valorAnalogicoV, PARADO_JOYSTICK_Y, MAX_JOYSTICK_Y, MIN_PWM, MAX_PWM);
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X não movimentado
  else {···
  }
}
```

Figura 27: Ré para a direita

- **Trás : Esquerda** - Se os *joysticks* vertical e horizontal forem pressionados para trás (+) e para a esquerda (-), respectivamente, o robô deve andar para trás e para a esquerda. Neste, o sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda esquerda deve ser menor que a velocidade da roda direita.

```
// Analógico Y movimentado para trás
if (valorAnalogicoV > (PARADO_JOYSTICK_Y + TOLERANCIA_JOYSTICK)) {
  pwmMotorDireito1 = MIN_PWM;
  pwmMotorEsquerdo2 = MIN_PWM;

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {
    pwmMotorDireito2 = map(valorAnalogicoV, PARADO_JOYSTICK_Y, MAX_JOYSTICK_Y, MIN_PWM, MAX_PWM);
    pwmMotorEsquerdo1 = map(valorAnalogicoV + valorAnalogicoH,
                            PARADO_JOYSTICK_Y + MIN_JOYSTICK_X,
                            MAX_JOYSTICK_Y + PARADO_JOYSTICK_X,
                            MIN_PWM, MAX_PWM);
  }

  // Analógico X não movimentado
  else {···
  }
}
```

Figura 28: Ré para a esquerda

- **Ré** - Se o *joystick* vertical for pressionado para trás (+) e o *joystick* horizontal não for pressionado (0), o robô deve andar para trás apenas. Para esse caso, a velocidade e o sentido de rotação de ambas as rodas devem ser iguais;

```
// Analógico Y movimentado para trás
if (valorAnalogicoV > (PARADO_JOYSTICK_Y + TOLERANCIA_JOYSTICK)) {
  pwmMotorDireito1 = MIN_PWM;
  pwmMotorEsquerdo2 = MIN_PWM;

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X não movimentado
  else {
    pwmMotorDireito2 = map(valorAnalogicoV, PARADO_JOYSTICK_Y, MAX_JOYSTICK_Y, MIN_PWM, MAX_PWM);
    pwmMotorEsquerdo1 = map(valorAnalogicoV, PARADO_JOYSTICK_Y, MAX_JOYSTICK_Y, MIN_PWM, MAX_PWM);
  }
}
```

Figura 29: Ré

- **Frente : Direita** - Se os *joysticks* vertical e horizontal forem pressionados para frente (-) e para a direita (+), respectivamente, o robô deve andar para frente e para a direita. Neste caso, o sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda direita deve ser menor que a da roda esquerda.

```
// Analógico Y movimentado para frente
else if (valorAnalogicoV < (PARADO_JOYSTICK_Y - TOLERANCIA_JOYSTICK)) {
  pwmMotorDireito2 = MIN_PWM;
  pwmMotorEsquerdo1 = MIN_PWM;

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {
    pwmMotorDireito1 = map(valorAnalogicoV + valorAnalogicoH,
                           MIN_JOYSTICK_Y + PARADO_JOYSTICK_X,
                           PARADO_JOYSTICK_Y + MAX_JOYSTICK_X,
                           MAX_PWM, MIN_PWM);
    pwmMotorEsquerdo2 = map(valorAnalogicoV, MIN_JOYSTICK_Y, PARADO_JOYSTICK_Y, MAX_PWM, MIN_PWM);
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X não movimentado
  else {···
  }
}
```

Figura 30: Frente para a direita

- **Frente : Esquerda** - Se os *joysticks* vertical e horizontal forem pressionados para frente (-) e para a esquerda (-), respectivamente, o robô deve andar para frente e para a esquerda. Para esse caso, o sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda esquerda deve ser menor que a velocidade da roda direita.

```
// Analógico Y movimentado para frente
else if (valorAnalogicoV < (PARADO_JOYSTICK_Y - TOLERANCIA_JOYSTICK)) {
  pwmMotorDireito2 = MIN_PWM;
  pwmMotorEsquerdo1 = MIN_PWM;

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {
    pwmMotorDireito1 = map(valorAnalogicoV, MIN_JOYSTICK_Y, PARADO_JOYSTICK_Y, MAX_PWM, MIN_PWM);
    pwmMotorEsquerdo2 = map(valorAnalogicoV - valorAnalogicoH,
                            MIN_JOYSTICK_Y - PARADO_JOYSTICK_X,
                            PARADO_JOYSTICK_Y - MIN_JOYSTICK_X,
                            MIN_PWM, MAX_PWM);
  }

  // Analógico X não movimentado
  else {···
  }
}
```

Figura 31: Frente para a esquerda

- **Frente** - Se o *joystick* vertical for pressionado para frente (-) e o *joystick* horizontal não for pressionado (0), o robô deve andar apenas para frente. Para esse caso, a velocidade e o sentido de rotação de ambas as rodas devem ser iguais;

```
// Analógico Y movimentado para frente
else if (valorAnalogicoV < (PARADO_JOYSTICK_Y - TOLERANCIA_JOYSTICK)) {
  pwmMotorDireito2 = MIN_PWM;
  pwmMotorEsquerdo1 = MIN_PWM;

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {···
  }

  // Analógico X não movimentado
  else {
    pwmMotorDireito1 = map(valorAnalogicoV, MIN_JOYSTICK_Y, PARADO_JOYSTICK_Y, MAX_PWM, MIN_PWM);
    pwmMotorEsquerdo2 = map(valorAnalogicoV, MIN_JOYSTICK_Y, PARADO_JOYSTICK_Y, MAX_PWM, MIN_PWM);
  }
}
```

Figura 32: Frente

- **Rotação horária no próprio eixo** - Se o *joystick* horizontal for pressionado para a direita (+) e o *joystick* vertical não for pressionado (0), o robô deve rotacionar, no próprio eixo, para o sentido horário. Neste caso, a velocidade de rotação de ambas as rodas deve ser igual, entretanto, a roda da direita deve rodar para trás e a da esquerda, para frente;
- **Rotação anti-horária no próprio eixo** - Se o *joystick* horizontal for pressionado para a esquerda (-) e o *joystick* vertical não for pressionado (nulo), o robô deve rotacionar, no próprio eixo, para o sentido anti-horário. Neste caso, a velocidade de rotação de ambas as rodas deve ser igual, entretanto, a roda da direita deve rodar para frente e a da esquerda, para trás;
- **Imobilidade** - Se nenhum *joystick* for pressionado, o robô permanece imóvel.

```
// Analógico Y não movimentado
else {

  // Analógico X movimentado para direita
  if (valorAnalogicoH > (PARADO_JOYSTICK_X + TOLERANCIA_JOYSTICK)) {
    pwmMotorDireito1 = MIN_PWM;
    pwmMotorEsquerdo1 = MIN_PWM;
    pwmMotorDireito2 = map(valorAnalogicoH, PARADO_JOYSTICK_X, MAX_JOYSTICK_X, MIN_PWM, MAX_PWM);
    pwmMotorEsquerdo2 = map(valorAnalogicoH, PARADO_JOYSTICK_X, MAX_JOYSTICK_X, MIN_PWM, MAX_PWM);
  }

  // Analógico X movimentado para esquerda
  else if (valorAnalogicoH < (PARADO_JOYSTICK_X - TOLERANCIA_JOYSTICK)) {
    pwmMotorDireito1 = map(valorAnalogicoH, MIN_JOYSTICK_X, PARADO_JOYSTICK_X, MAX_PWM, MIN_PWM);
    pwmMotorEsquerdo1 = map(valorAnalogicoH, MIN_JOYSTICK_X, PARADO_JOYSTICK_X, MAX_PWM, MIN_PWM);
    pwmMotorDireito2 = MIN_PWM;
    pwmMotorEsquerdo2 = MIN_PWM;
  }

  // Analógico X não movimentado
  else {
    pwmMotorDireito1 = MIN_PWM;
    pwmMotorEsquerdo1 = MIN_PWM;
    pwmMotorDireito2 = MIN_PWM;
    pwmMotorEsquerdo2 = MIN_PWM;
  }
}
```

Figura 33: Rotação no próprio eixo e imobilidade

Para os casos em que uma roda deve rotacionar com velocidade menor do que a outra, de modo que o robô consiga se movimentar simultaneamente para a esquerda ou direita e para frente ou para trás, utiliza-se uma combinação (+ ou -) dos valores vertical e horizontal para encontrar o valor PWM para a roda mais lenta, e apenas o valor vertical para a roda mais rápida. Já para os casos em que ambas as rodas movem-se com velocidades iguais, utiliza-se apenas o valor vertical ou o valor horizontal para determinar o valor PWM.

Conforme visto na Apostila de Hardware, a saída analógica da ESP32 é capaz de gerar valores que vão de 0 a 255. Porém, a combinação dos valores dos *joysticks* pode gerar resultados que vão de -1024 a 1024 nos casos mais extremos. Portanto, é necessário converter o valor da combinação para um valor entre 0 e 255. Para tal, o comando map() foi utilizado.

A função map() recebe como parâmetro o valor que se deseja converter, o limite inferior do intervalo de origem do valor, o limite superior do intervalo de origem do valor, o limite inferior do intervalo de destino do valor e o limite superior do intervalo de destino do valor, nessa ordem. A partir dessas informações, ele calcula um novo valor dentro do intervalo de destino.

$$= map(valor, deMenor, deMaior, paraMenor, paraMaior); \quad (1)$$

1. valor: o número a ser mapeado
2. deMenor: o menor limite do intervalo atual do valor
3. deMaior: o maior limite do intervalo atual do valor
4. paraMenor: o menor limite do intervalo alvo
5. paraMaior: o maior limite do intervalo alvo

Para cada possível combinação dos *joysticks* (frente + direita, trás + esquerda, ...) existe uma faixa de valores para o intervalo de origem (atual). Entretanto, o intervalo de saída será sempre de 0 a 255 ou de 255 a 0. O código garante que um pino do motor receba um valor baixo de PWM e que o outro receba um valor alto, produzindo assim uma diferença de potência e fazendo com que o motor rotacione. Quanto maior for a diferença entre o PWM de cada pino, maior a rotação do motor.

PARADO_JOYSTICK_Y e **PARADO_JOYSTICK_X** representam os valores que os *joysticks* retornam na vertical e na horizontal, respectivamente, quando não são movimentados.

MAX_JOYSTICK_Y e **MIN_JOYSTICK_Y** representam, respectivamente, o maior e o menor valor que ambos os *joysticks* retornam quando verticalmente movimentados (valor quando o joystick é inteiramente pressionado para trás e para frente).

MAX_JOYSTICK_X e **MIN_JOYSTICK_X** representam, respectivamente, o maior e o menor valor que ambos os *joysticks* retornam quando horizontalmente movimentados (valor quando o joystick é inteiramente pressionado para a direita e para a esquerda).

Esses seis valores variam de controle para controle e podem ser obtidos utilizando um código que será demonstrado mais adiante.

As variáveis **MIN_PWM** e **MAX_PWM** armazenam os limites da faixa PWM para os quais as combinações dos *joysticks* devem ser convertidas.

Foi definida uma **zona morta para o controle**, ou seja, se os *joysticks* forem levemente pressionados, de forma que os valores lidos sejam menores do que a tolerância estipulada, o programa desconsidera o valor lido. Isso foi feito para **evitar que o robô fique se movimentando sozinho**, devido às vibrações mecânicas no controle e afins.

3.6.5 Inversão de rotação dos motores

A lógica de funcionamento da locomoção do robô, descrita no início do subtópico acima, só é válida se os terminais dos motores forem corretamente conectados à placa eletrônica. Para resolver esse problema, projetou-se um trecho de código que permite a troca dos pinos de cada motor. A troca é feita pelas setinhas do controle. Cada setinha representa uma possível combinação de pinos.

```
/* ----------------- Lógica de inversão de giro da movimentação ----------------- */

uint8_t leituraSetinhas = myController->dpad();

// Cada SETINHA representa uma configuração de pinos para os motores
switch (leituraSetinhas) {
  case 0x01: // cima
    sentidoMotorEsquerdo = PINO_1_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_2_MOTOR_ESQUERDO;
    sentidoMotorDireito = PINO_1_MOTOR_DIREITO, velocidadeMotorDireito = PINO_2_MOTOR_DIREITO;
    break;
  case 0x02: // baixo
    sentidoMotorEsquerdo = PINO_1_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_2_MOTOR_ESQUERDO;
    sentidoMotorDireito = PINO_2_MOTOR_DIREITO, velocidadeMotorDireito = PINO_1_MOTOR_DIREITO;
    break;
  case 0x04: // direita
    sentidoMotorEsquerdo = PINO_2_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_1_MOTOR_ESQUERDO;
    sentidoMotorDireito = PINO_1_MOTOR_DIREITO, velocidadeMotorDireito = PINO_2_MOTOR_DIREITO;
    break;
  case 0x08: // esquerda
    sentidoMotorEsquerdo = PINO_2_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_1_MOTOR_ESQUERDO;
    sentidoMotorDireito = PINO_2_MOTOR_DIREITO, velocidadeMotorDireito = PINO_1_MOTOR_DIREITO;
    break;
}
```

Figura 34: Trecho para inversão de rotação dos motores

Por padrão, a configuração da setinha esquerda é assumida sempre que o robô é energizado. Uma vez modificada, o controle permanecerá com a configuração definida. Entretanto, sempre que o robô é desligado, a configuração é perdida, sendo necessário defini-la novamente ao ligá-lo.

3.7 Função setup()

A função setup() é própria da ESP32 e, portanto, é *obrigatória* para o funcionamento do código. Ela é executada, de forma automática, uma única vez durante todo o funcionamento do programa, no momento em que o controlador é energizado.

De forma geral, neste trecho, colocamos todas as configurações de funcionamento da ESP32 (definição dos pinos de entrada e saída, inicialização da serial, ...), inicializamos as bibliotecas utilizadas e definimos os estados iniciais das variáveis.

No código utilizado, primeiramente, inicializa-se a comunicação serial. Ela é utilizada para exibir as mensagens de funcionamento no Monitor Serial, facilitando o entendimento do que está acontecendo. Em sequência, inicializa-se a biblioteca Bluepad32 (observe que as funções de conexão e desconexão do controle são passadas como parâmetro durante a inicialização) e removem-se todos os dispositivos que haviam sido pareados anteriormente. Depois, definem-se os pinos nos quais as pontes H estão conectadas como saída (OUTPUT), utilizando o comando pinMode, e assume-se uma configuração de rotação inicial para os motores de locomoção (equivalente à configuração setinha para a esquerda). Por fim, garante-se que todos os motores estejam desligados (desligaRobo()).

```
void setup() {

  Serial.begin(115200);

  // Inicia comunicação Bluetooth que permite a conexão com controle.
  BP32.setup(&onConnectedController, &onDisconnectedController);
  BP32.enableVirtualDevice(false);

  // Desparea os controles que haviam sido conectados anteriormente.
  BP32.forgetBluetoothKeys();

  // Configura pinos da ESP32 para controle dos motores de arma.
  pinMode(PINO_1_ARMA1, OUTPUT);
  pinMode(PINO_2_ARMA1, OUTPUT);

  pinMode(PINO_1_ARMA2, OUTPUT);
  pinMode(PINO_2_ARMA2, OUTPUT);

  // Configura pinos da ESP32 para controle dos motores de locomoção.
  pinMode(PINO_1_MOTOR_ESQUERDO, OUTPUT);
  pinMode(PINO_2_MOTOR_ESQUERDO, OUTPUT);

  pinMode(PINO_1_MOTOR_DIREITO, OUTPUT);
  pinMode(PINO_2_MOTOR_DIREITO, OUTPUT);

  sentidoMotorEsquerdo = PINO_2_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_1_MOTOR_ESQUERDO;
  sentidoMotorDireito = PINO_2_MOTOR_DIREITO, velocidadeMotorDireito = PINO_1_MOTOR_DIREITO;

  direitoVesquerdoH = true, direitoHesquerdoV = false;

  // Desliga movimentação e arma do robô.
  desligaRobo();
}
```

Figura 35: setup()

3.8 Função loop()

Assim que o setup() é executado, a ESP32 passa automaticamente a executar a função loop(). Ela é executada de forma intermitente, ou seja, o controlador executa tudo que está dentro dela; uma vez terminada a execução, volta para o início e executa novamente. Esse processo se repete até que a ESP32 seja desligada. Ela também é própria do controlador e, portanto, é **obrigatória** para o funcionamento do código.

A função loop() contida no código é bastante reduzida; todo o descritivo do funcionamento do robô está presente na função processControllers(). Neste trecho, há apenas uma verificação de atualização nos dados do controle. Em caso afirmativo, executa-se a função processControllers() para processar os novos dados.

```
void loop() {
  // Checa se houve atualização nos dados do controle
  bool dataUpdated = BP32.update();

  // Se sim, chama a função processControllers() para processar os dados
  if (dataUpdated)
    processControllers();
}
```

Figura 36: loop()

4 Arquivo parametros.h

Na mesma pasta do código principal há um arquivo de configurações, denominado parametros.h (.header). Esse arquivo foi criado com o intuito de deixar o código mais organizado, facilitando a alteração de alguns parâmetros caso necessário.

Nas duas primeiras linhas há a inicialização do arquivo. Se ele não estiver sido definido (if not defined - #ifndef), o definimos (#define) como sendo parametros_h. Pode ser que existam arquivos de configuração com o mesmo nome, por isso esses comandos se fazem necessários. Perceba que o nome na definição deve ser semelhante ao nome do arquivo, diferindo apenas no ponto/underline. O comando #endif encerra o arquivo.

Em sequência, há a inclusão da biblioteca Arduino.h, necessária para que se possa programar na IDE do Arduino e para que alguns comandos sejam compreendidos corretamente.

codigo_robo_controle_p3.ino | parametros.h

```
1
2   #ifndef parametros_h
3   #define parametros_h
4
5   #include <Arduino.h>
```

Figura 37: Inicialização do arquivo .h

Para entender um pouco mais sobre os comandos #ifndef, #endif, entre outros, chamados de diretivas, clique **aqui**.

Logo abaixo vem a definição dos pinos utilizados para controlar os motores de movimentação e de arma. Digitar no código principal o nome escrito após o comando #define é o mesmo que digitar o valor numérico que está do lado direito do nome. Essa definição é importante pois facilita a identificação dos pinos no código principal.

```
7   #define PINO_1_ARMA2 13
8   #define PINO_2_ARMA2 12
9
10  #define PINO_1_MOTOR_ESQUERDO 14
11  #define PINO_2_MOTOR_ESQUERDO 27
12
13  #define PINO_1_MOTOR_DIREITO 25
14  #define PINO_2_MOTOR_DIREITO 26
15
16  #define PINO_1_ARMA1 33
17  #define PINO_2_ARMA1 32
```

Figura 38: Defines pinos motores

Sobre os dados salvos no arquivo, primeiramente vem os parâmetros dos *joysticks* do controle: valores centrais, máximos e mínimos. Em sequência, o valor da zona morta, ou seja, a partir de qual valor deseja-se que o código passe a considerar a leitura do analógico como sendo diferente de 0 (utilizado para evitar trepidações). Por último há a definição dos limites da faixa de valores que será utilizada para controlar as pontes H. É valido lembrar que estes valores estão relacionados à resolução da saída analógica (PWM) da ESP32.

Para o controle utilizado, os valores retornados por ambos os *joysticks*, para uma mesma direção e sentido, são idênticos (ex.: quando ambos são prescionados para a esquerda, retornam o mesmo valor). Além disso, ambas as direções possuem a mesma faixa de retorno de valores, conforme demonstrado na Figura 39 abaixo.

```
const int32_t MIN_JOYSTICK_Y = -512, PARADO_JOYSTICK_Y = 0, MAX_JOYSTICK_Y = 508;
const int32_t MIN_JOYSTICK_X = -512, PARADO_JOYSTICK_X = 0, MAX_JOYSTICK_X = 508;

const int32_t TOLERANCIA_JOYSTICK = 5;  // zona morta do controle

const int MIN_PWM = 0, MAX_PWM = 255;  // valores limite para o PWM da ESP32
```

Figura 39: Definição de parâmetros

A palavra *const* no início da definição dos parâmetros indica que eles não mudarão ao longo da execução do código, terão sempre o valor definido no arquivo .h. Sempre que uma dessas variáveis for escrita no código principal ela estará referenciando o valor descrito nesse arquivo.