

Apostila de Firmware

2025



Índice

1	Instalações	3
1.1	Arduino IDE	3
1.2	Driver CP210x	9
1.3	Biblioteca Bluepad32	12
2	Linguagem Arduino	17
3	Código principal do robô	17
3.1	Importando Bibliotecas	17
3.2	Definição de variáveis	17
3.3	Função desligaRobo()	18
3.4	Função onConnectedController()	19
3.5	Função onDisconnectedController()	19
3.6	Função processControllers()	20
3.6.1	Verificação de conexão	20
3.6.2	Liga/Desliga robô	20
3.6.3	Arma	21
3.6.4	Movimentação	22
3.6.5	Inversão de rotação dos motores	27
3.7	Função setup()	28
3.8	Função loop()	29
4	Arquivo parametros.h	30




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 e ESP32. 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;
<https://www.arduino.cc/en/software>
2. Selecione o sistema operacional utilizado pelo seu computador. Para computadores Windows, selecione a primeira opção, conforme demonstrado na Figura 1 abaixo. Ao clicar em uma das opções, você será redirecionado para outra página;

Downloads



Arduino IDE 2.3.4

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits

Windows MSI installer

Windows ZIP file

Linux AppImage 64 bits (X86-64)

Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits

macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

Figure 1: Página de seleção do sistema operacional

3. Selecione a opção JUST DOWNLOAD. Você será redirecionado para outra página;

Download Arduino IDE & support its progress

Since the 1.x release in March 2015, the Arduino IDE has been downloaded **94.392.570** times — impressive! Help its development with a donation.

\$3

\$5

\$10

\$25

\$50

Other

CONTRIBUTE AND DOWNLOAD

or

JUST DOWNLOAD



Learn more about [donating to Arduino](#).

Figure 2: Página Just Download 1

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



Stay in the Loop: Join Our Newsletter!

As a beginner or advanced user, you can find inspiring projects and learn about cutting-edge Arduino products through our **weekly newsletter!**

email *

☐

I confirm to have read the [Privacy Policy](#) and to accept the [Terms of Service](#) *

☐

I would like to receive emails about special deals and commercial offers from Arduino.

SUBSCRIBE & DOWNLOAD

or

JUST DOWNLOAD



Figure 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;

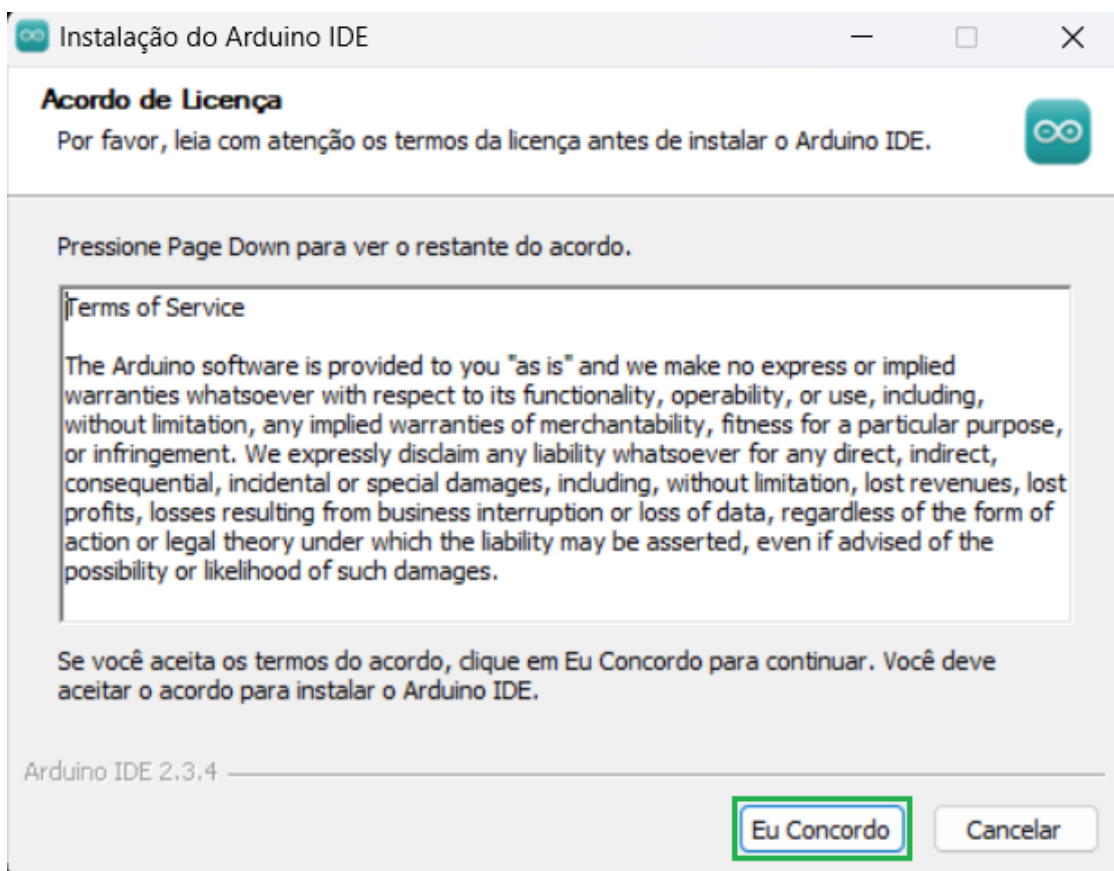


Figure 4: Concordância com os Termos de Licença

7. Em sequência, selecione para quais usuários de seu computador o aplicativo deve ser instalado. Recomendamos instalar apenas para seu usuário. Feita a seleção, clique em Próximo;

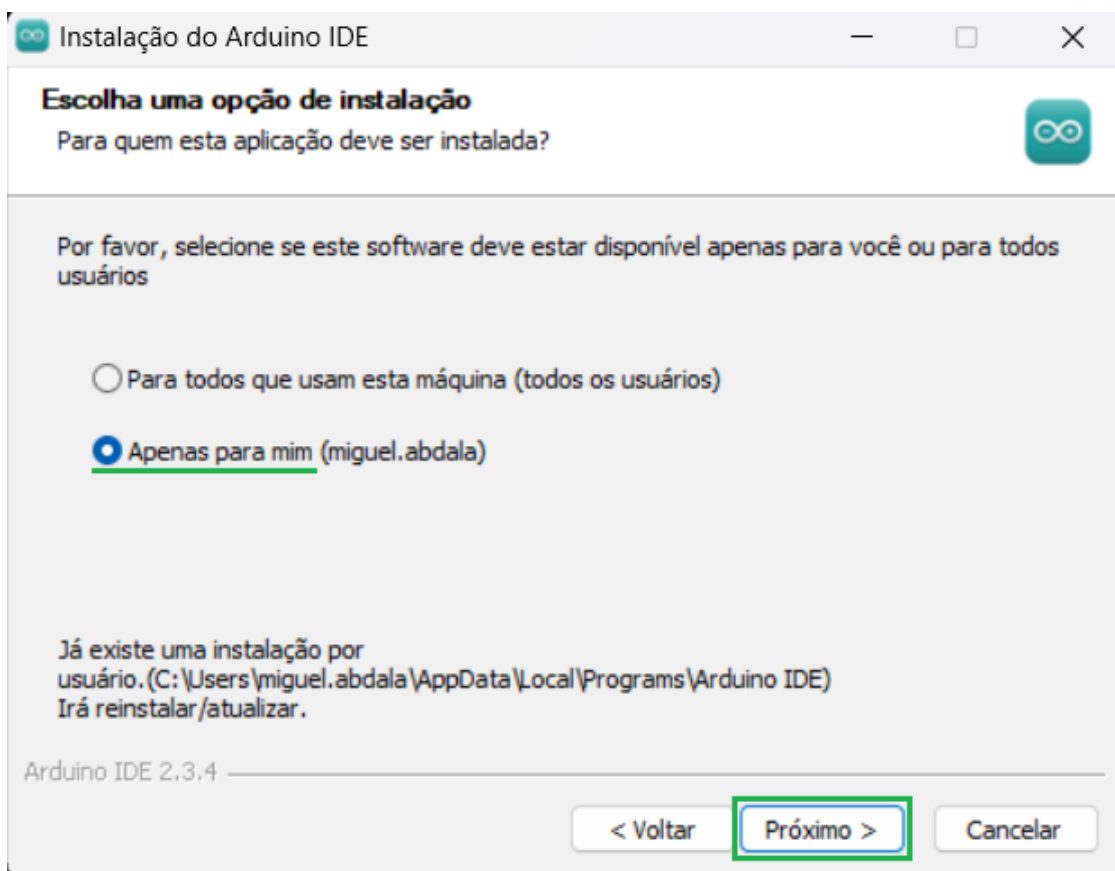


Figure 5: Opções de instalação

- Escolha o local de instalação do programa. Recomendamos deixar a Pasta de Destino padrão escolhida pelo próprio instalador, para evitar possíveis problemas futuros. Após, clique em Instalar para começar a instalação;

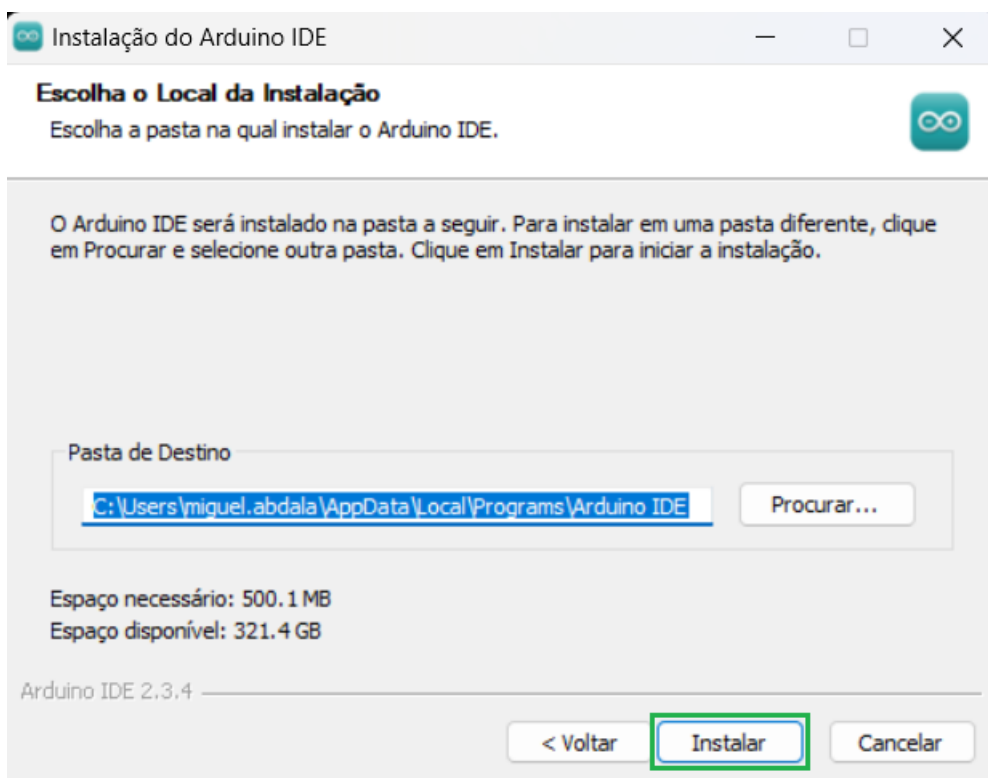


Figure 6: Seleção do Local de Instalação do programa

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



Figure 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 ele 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
<https://www.silabs.com/developer-tools/usb-to-uart-bridge-vcp-drivers?tab=downloads>
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;



Software Downloads

Software (11)

Software · 11

CP210x Universal Windows Driver	v11.4.0 12/18/2024
CP210x VCP Mac OSX Driver	v6.0.2 10/26/2021
CP210x VCP Windows	v6.7 9/3/2020
CP210x Windows Drivers	v6.7.6 9/3/2020
CP210x Windows Drivers with Serial Enumerator	v6.7.6 9/3/2020

[Show 6 more Software](#)

Figure 8: Download 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, basta clicar sobre ele com o botão direito do mouse e depois clicar em extrair;
4. Abra a pasta originada a partir da extrassão do arquivo .zip, clique no arquivo de nome silabser com o botão direito e clique em instalar;

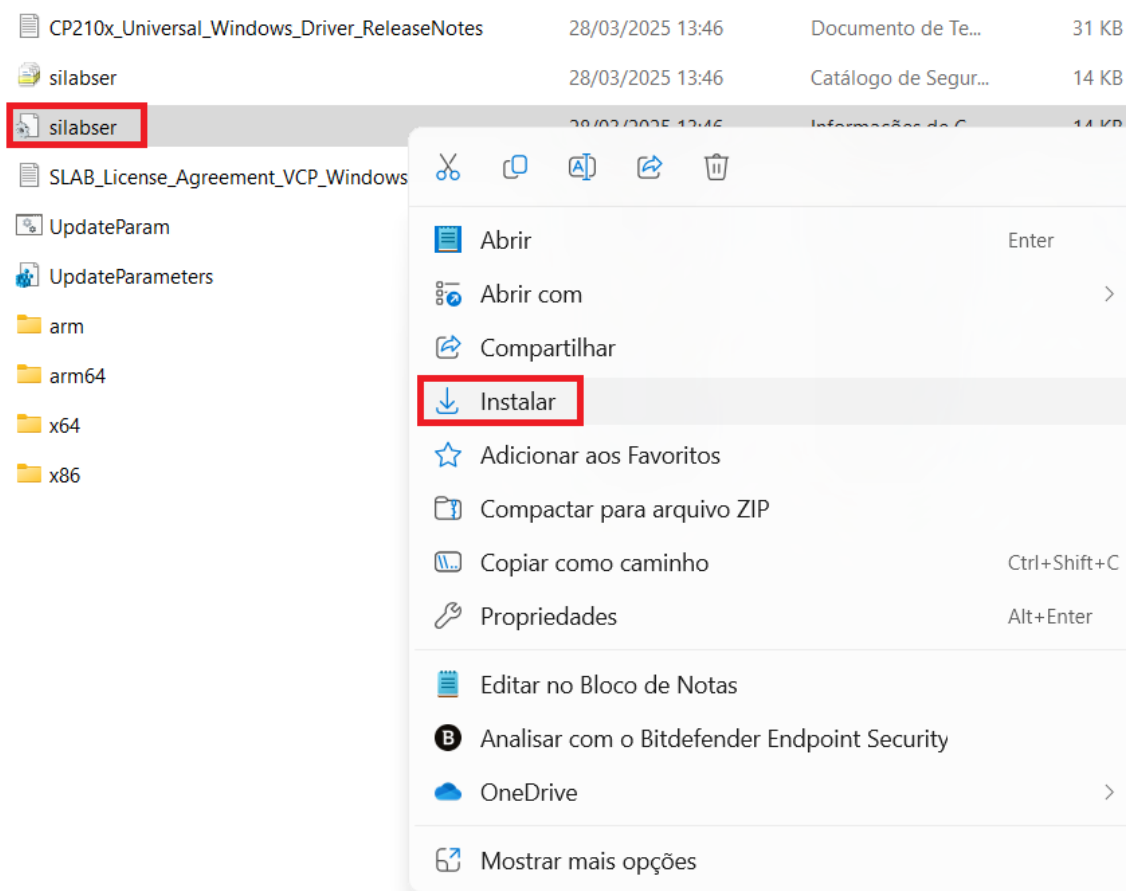


Figure 9: Iniciando instalação do driver

5. Na nova janela que se abrirá, clique em Abrir. Caso uma janela de notificação do Windows seja aberta logo em sequência, pressione Sim;

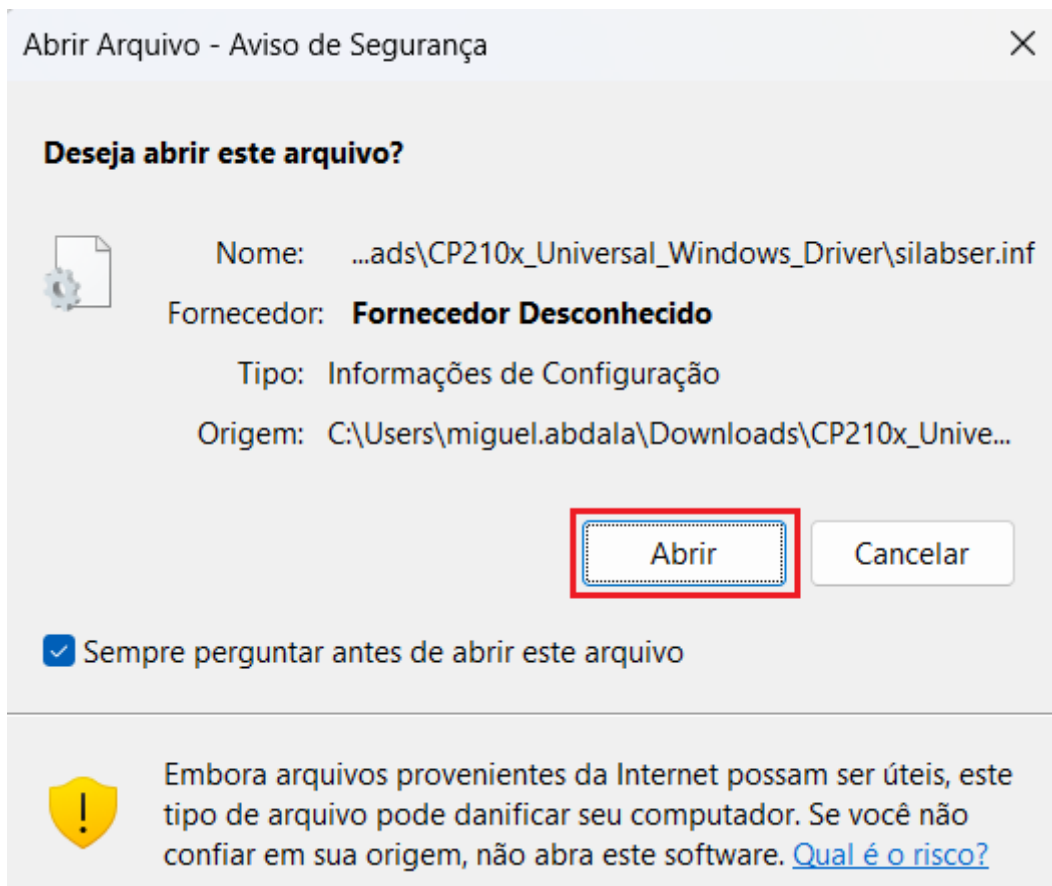


Figure 10: Instalando driver

6. Feito isso, o driver estará instalado. Clique em Ok para finalizar o processo.

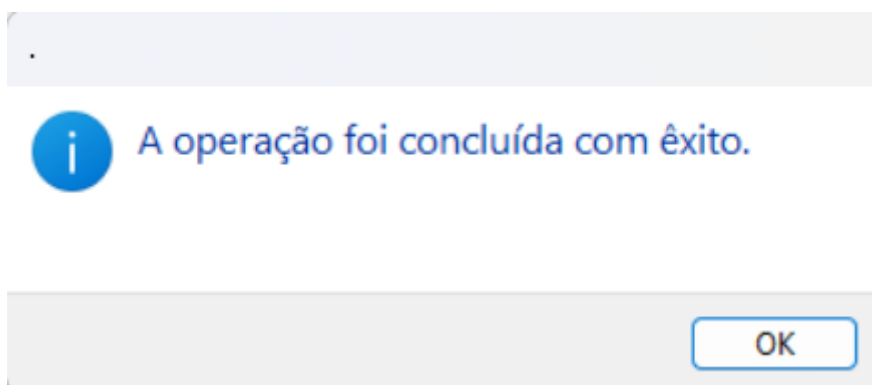


Figure 11: Driver instalado com sucesso

1.3 Biblioteca Bluepad32

Bibliotecas, no contexto de programação, são conjuntos de códigos prontos que facilitam a criação de projetos. Elas funcionam como "caixinhas de ferramentas" que adicionam novas funções ao microcontrolador, 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.

Link da documentação oficial da biblioteca <https://bluepad32.readthedocs.io/en/latest/>

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

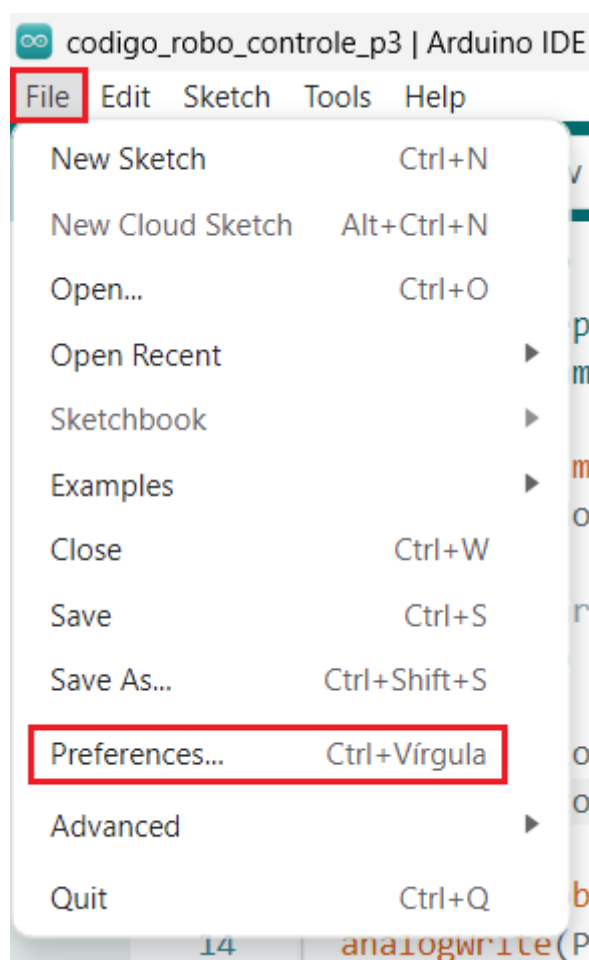


Figure 12: Abrindo campo de Preferências da IDE

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

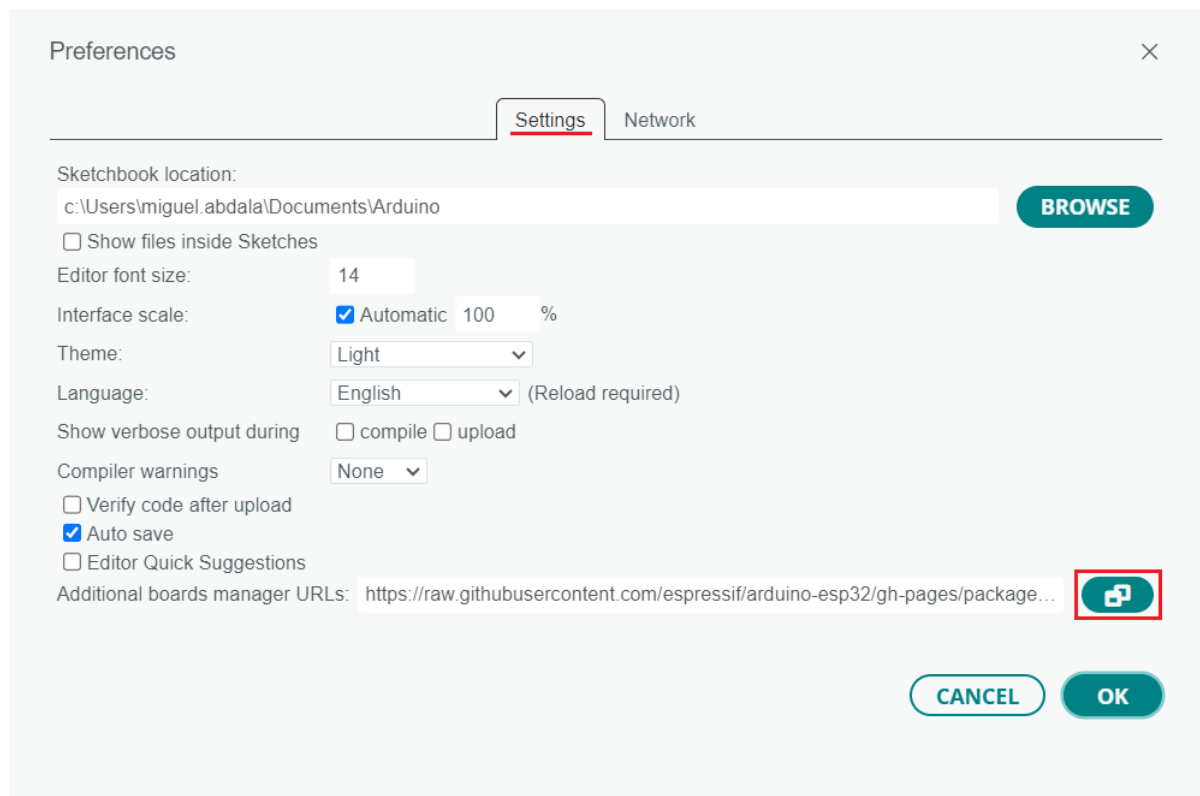


Figure 13: Janela de Preferências

3. Cole os links abaixo no campo que aparecer, um em baixo 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

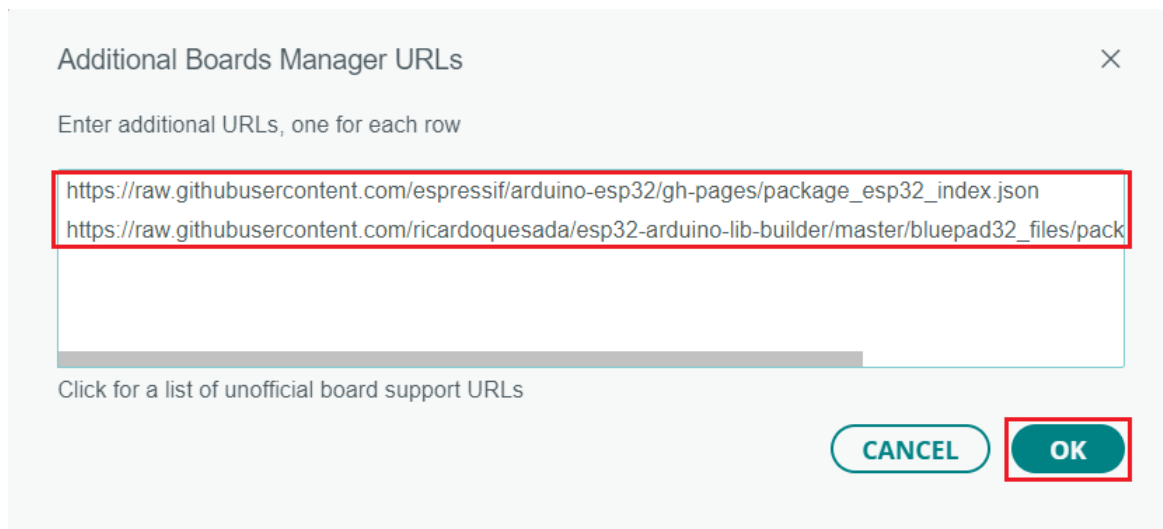


Figure 14: Identificação dos locais de download da biblioteca

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

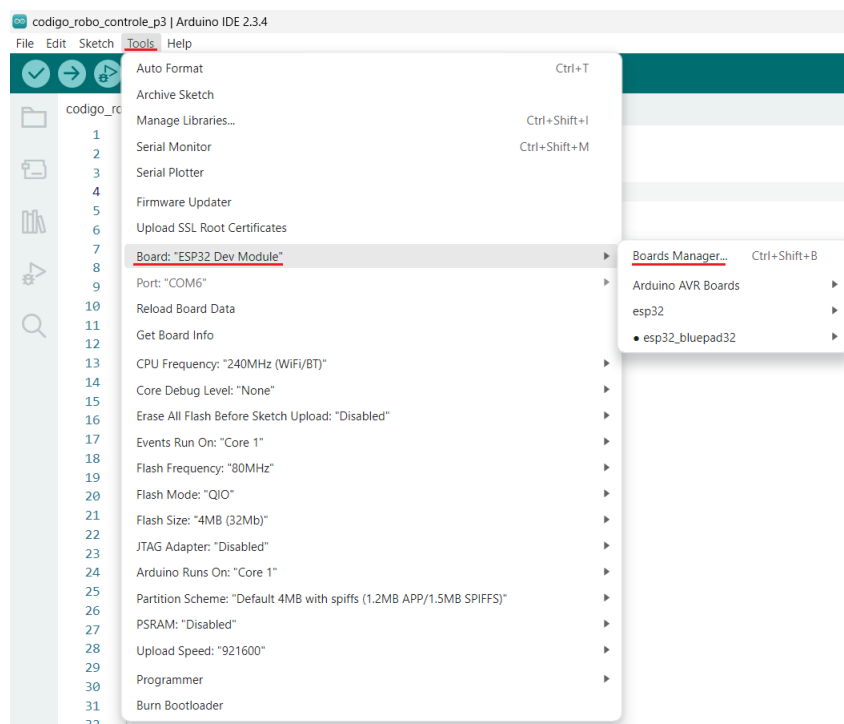


Figure 15: Abrindo o gerenciador de placas

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;*

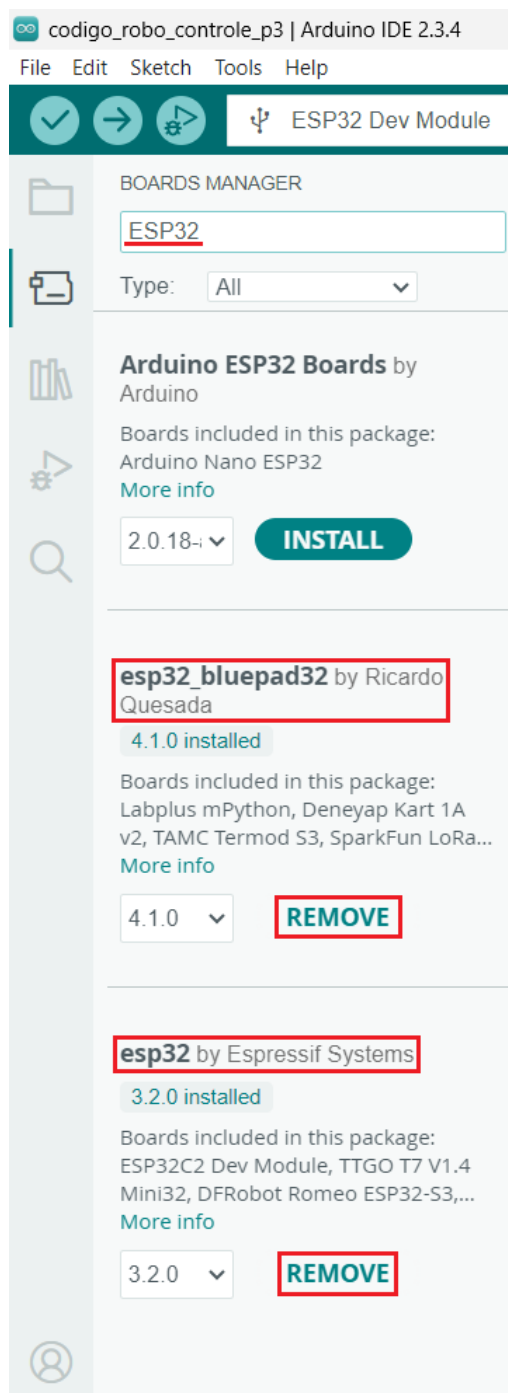


Figure 16: Instalando placas

Observações:

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



2 Linguagem Arduino

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

- Variáveis: <https://docs.arduino.cc/language-reference/pt/#vari%C3%A1veis>
- Estruturas: <https://docs.arduino.cc/language-reference/pt/#estruturas>
- Funções: <https://docs.arduino.cc/language-reference/pt/#fun%C3%A7%C3%B5es>

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"
```

Figure 17: Importando biblioteca e arquivo de configurações

3.2 Definição de variáveis

Após as inclusões, seis variáveis que serão utilizadas ao longo do programa são declaradas. Abaixo consta um 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: utilizadas durante a execução do código para indicar os pinos nos quais os terminais do motor esquerdo do robô estão conectados;
- sentidoMotorDireito e velocidadeMotorDireito: utilizadas durante a execução do código para indicar os pinos nos quais os terminais do motor direito do robô estão conectados;

```
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;
```

Figure 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) às saídas associadas ao controle dos motores de arma e de locomoção (todos os motores são configurados com velocidade 0). Ao final, defini-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 }
```

Figure 19: Função desligaRobo()



3.4 Função onConnectedController()

Responsável por estabelecer a conexão entre o controle e a ESP32. 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). É importante ressaltar que a execução dessa função é feita de forma automática pela ESP32, no momento de pareamento do controle.

```
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 controle for estabelecida, 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 }
```

Figure 20: Função onConnectedController()

3.5 Função onDisconnectedController()

Responsável por desconectar o controle da ESP32. 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). É importante ressaltar que a execução dessa função é feita de forma automática pela ESP32, no momento da desconexão do controle.



```
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 }
```

Figure 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 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 executadas 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. */
        }
    }
}
```

Figure 22: Verificação de conexão

3.6.2 Liga/Desliga robô

Feita a verificação da conexão, lê-se o estado dos botões SELECT e START. Se SELECT for pressionado (estadoMiscButtons == 0x02), o robô é desligado (roboLigado = false). Caso contrário, se START for pressionado (estadoMiscButtons == 0x04), liga-se o robô (roboLigado = true).



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

```
uint16_t estadoMiscButtons = myController->miscButtons();

// Se SELECT for pressionado, desliga robô
if (estadoMiscButtons == 0x02) {
    roboLigado = false;
    Serial.println("Robo desligado");
}

// Se START for pressionado, liga robô
else if (estadoMiscButtons == 0x04) {
    roboLigado = true;
    Serial.println("Robo iniciado");
}
```

Figure 23: Liga/Desliga robô

3.6.3 Arma

Os motores de arma são **simultaneamente** comandados pelos botões descritos abaixo

- TRIÂNGULO - botoesPressionados == 0x0008: desliga motores
- BOLINHA - botoesPressionados == 0x0002: liga motores em um sentido
- QUADRADO - botoesPressionados == 0x0004: liga motores em outro sentido

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



```
90     if (roboLigado) {
91
92         /* ----- Lógica de funcionamento e inversão de giro da arma ----- */
93
94         uint16_t botoesPressionados = myController->buttons();
95
96         // Se TRIÂNGULO for pressionado, desliga motores da arma.
97         if (botoesPressionados == 0x0008) {
98             Serial.print("Arma desligada\n");
99             analogWrite(PINO_1_ARMA1, 0);
100             analogWrite(PINO_2_ARMA1, 0);
101             analogWrite(PINO_1_ARMA2, 0);
102             analogWrite(PINO_2_ARMA2, 0);
103         }
104
105         // Se BOLINHA for pressionado, roda arma para um lado
106         if (botoesPressionados == 0x0002) {
107             Serial.print("Arma Sentido 1\n");
108             analogWrite(PINO_1_ARMA1, 0);
109             analogWrite(PINO_2_ARMA1, maxPWM);
110             analogWrite(PINO_1_ARMA2, maxPWM); // para rodar pro outro lado
111             analogWrite(PINO_2_ARMA2, 0);
112         }
113
114         // Se QUADRADO for pressionado, roda arma para o outro lado
115         else if (botoesPressionados == 0x0004) {
116             Serial.print("Arma Sentido 2\n");
117             analogWrite(PINO_1_ARMA1, maxPWM);
118             analogWrite(PINO_2_ARMA1, 0);
119             analogWrite(PINO_1_ARMA2, 0); // para rodar pro outro lado
120             analogWrite(PINO_2_ARMA2, maxPWM);
121         }
122     }
```

Figure 24: Controle dos motores de arma

3.6.4 Movimentação

Quando movimentados, os analógicos retornam um valor proporcional ao quão pressionado estão em cada direção (vertical ou horizontal). De acordo com a documentação da Bluepad32, para ambas direções (horizontal e vertical), o valor retornado pode variar de -511 a 512. O sinal do valor depende da direção de pressão. De maneira geral, 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 são controlados pelo movimento vertical do analógico direito e o movimento horizontal do analógico esquerdo. O valor de velocidade colocado em cada roda é proporcional à combinação desses dois valores ($\text{valorAnalogicoDireito} \pm \text{valorAnalogicoEsquerdo}$).



```
129      /* ----- Lógica de funcionamento da movimentação ----- */
130
131      // Lê valor em Y do analógico direito (R-right).
132      int32_t valorAnalogicoDireito = -myController->axisRY();
133
134      // Lê valor em X do analógico esquerdo (L-left).
135      int32_t valorAnalogicoEsquerdo = myController->axisX();
136
137      // Exibe valores no monitor serial.
138      Serial.print("Y analogico R: ");
139      Serial.println(valorAnalogicoDireito);
140
141      Serial.print("X analogico L: ");
142      Serial.println(valorAnalogicoEsquerdo);
143
144      int pwmMotorDireito1, pwmMotorDireito2, pwmMotorEsquerdo1, pwmMotorEsquerdo2;
```

Figure 25: Leitura dos analógicos

O valor vertical do analógico direito está sendo multiplicado por -1 para deixar no mesmo sentido do plano cartesiano (cima +, baixo -), facilitando a compreensão.

As variáveis pwmMotorDireito1, pwmMotorDireito2, pwmMotorEsquerdo1 e pwmMotorEsquerdo2 servem para controlar a velocidade de rotação dos motores.

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

- **Frente/Direita** - Se o analógico direito for pressionado para frente (+) e o analógico esquerdo for pressionado para a direita (+), o robô deve andar para frente e para direita. Para esse caso, a sentido de rotação de ambas rodas deve ser igual. Entretanto, a velocidade da roda esquerda deve ser maior que a velocidade da roda direita, para que o robô vire para a direita.

```
146      if (valorAnalogicoDireito > (centerAnalogR_Y + toleranciaAnalogico)) {
147
148          pwmMotorDireito1 = maxPWM;
149          pwmMotorEsquerdo1 = maxPWM;
150
151          if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) {
152
153              pwmMotorDireito2 = map(valorAnalogicoDireito - valorAnalogicoEsquerdo,
154                                     centerAnalogR_Y - maxAnalogL_X,
155                                     maxAnalogR_Y - centerAnalogL_X,
156                                     maxPWM,
157                                     minPWM);
158
159              pwmMotorEsquerdo2 = map(valorAnalogicoDireito + valorAnalogicoEsquerdo,
160                                      centerAnalogR_Y + centerAnalogL_X,
161                                      maxAnalogR_Y + maxAnalogL_X,
162                                      maxPWM,
163                                      minPWM);
164          }
165
166      > else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) { ...
179      }
180
181      > else { ...
184      }
185  }
```

Figure 26: Frente para direita



- **Frente/Esquerda** - Se o analógico direito for pressionado para frente (+) e o analógico esquerdo for pressionado para a esquerda (-), o robô deve andar para frente e para esquerda. Para esse caso, a sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda direita deve ser maior que a velocidade da roda esquerda, para que o robô vire para a esquerda.

```
146 |         if (valorAnalogicoDireito > (centerAnalogR_Y + toleranciaAnalogico)) {
147 |             pwmMotorDireito1 = maxPWM;
148 |             pwmMotorEsquerdo1 = maxPWM;
149 |
150 | >         if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) { ...
164 |             }
165 |
166 |             else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) {
167 |
168 |                 pwmMotorDireito2 = map(valorAnalogicoDireito - valorAnalogicoEsquerdo,
169 |                                         centerAnalogR_Y - centerAnalogL_X,
170 |                                         maxAnalogR_Y - minAnalogL_X,
171 |                                         maxPWM,
172 |                                         minPWM);
173 |
174 |                 pwmMotorEsquerdo2 = map(valorAnalogicoDireito + valorAnalogicoEsquerdo,
175 |                                         centerAnalogR_Y + minAnalogL_X,
176 |                                         maxAnalogR_Y + centerAnalogL_X,
177 |                                         maxPWM,
178 |                                         minPWM);
179 |             }
180 |
181 | >         else { ...
184 |             }
185 |     }
```

Figure 27: Frente para esquerda

- **Frente** - Se o analógico direito for pressionado para frente (+) e o analógico esquerdo não for pressionado (nulo), o robô deve andar apenas para frente. Para esse caso, a velocidade e o sentido de rotação de ambas as rodas devem ser iguais;

```
146 |         if (valorAnalogicoDireito > (centerAnalogR_Y + toleranciaAnalogico)) {
147 |             pwmMotorDireito1 = maxPWM;
148 |             pwmMotorEsquerdo1 = maxPWM;
149 |
150 | >         if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) { ...
164 |             }
165 |
166 | >         else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) { ...
179 |             }
180 |
181 |         else {
182 |             pwmMotorDireito2 = map(valorAnalogicoDireito, centerAnalogR_Y, maxAnalogR_Y, maxPWM, minPWM);
183 |             pwmMotorEsquerdo2 = map(valorAnalogicoDireito, centerAnalogR_Y, maxAnalogR_Y, maxPWM, minPWM);
184 |         }
185 |     }
```

Figure 28: Frente

- **Ré/Direita** - Se o analógico direito for pressionado para trás (-) e o analógico esquerdo for pressionado para a direita (+), o robô deve andar para trás e para direita. Para esse caso, a sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda esquerda deve ser maior que a velocidade da roda direita, para que o robô vire para a direita.



```
187 | else if (valorAnalogicoDireito < (centerAnalogR_Y - toleranciaAnalogico)) {
188 |
189 |     pwmMotorDireito1 = minPWM;
190 |     pwmMotorEsquerdo1 = minPWM;
191 |
192 |     if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) {
193 |         pwmMotorDireito2 = map(valorAnalogicoEsquerdo + valorAnalogicoDireito,
194 |                                centerAnalogL_X + minAnalogL_X,
195 |                                maxAnalogL_X + centerAnalogR_Y,
196 |                                minPWM,
197 |                                maxPWM);
198 |
199 |         pwmMotorEsquerdo2 = map(valorAnalogicoEsquerdo - valorAnalogicoDireito,
200 |                                centerAnalogL_X - centerAnalogR_Y,
201 |                                maxAnalogL_X - minAnalogR_Y,
202 |                                minPWM,
203 |                                maxPWM);
204 |     }
205 |
206 | > else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) { ...
218 | }
219 |
220 | > else { ...
223 | }
224 | }
```

Figure 29: Ré para direita

- **Ré/Esquerda** - Se o analógico direito for pressionado para trás (-) e o analógico esquerdo for pressionado para a esquerda (-), o robô deve andar para trás e para esquerda. Para esse caso, a sentido de rotação de ambas as rodas deve ser igual. Entretanto, a velocidade da roda direita deve ser maior que a velocidade da roda esquerda, para que o robô vire para a esquerda.

```
187 | else if (valorAnalogicoDireito < (centerAnalogR_Y - toleranciaAnalogico)) {
188 |
189 |     pwmMotorDireito1 = minPWM;
190 |     pwmMotorEsquerdo1 = minPWM;
191 |
192 | > if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) { ...
204 | }
205 |
206 |     else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) {
207 |         pwmMotorDireito2 = map(valorAnalogicoEsquerdo + valorAnalogicoDireito,
208 |                                centerAnalogL_X + centerAnalogR_Y,
209 |                                minAnalogL_X + minAnalogR_Y,
210 |                                minPWM,
211 |                                maxPWM);
212 |
213 |         pwmMotorEsquerdo2 = map(valorAnalogicoEsquerdo - valorAnalogicoDireito,
214 |                                minAnalogL_X - centerAnalogR_Y,
215 |                                centerAnalogL_X - minAnalogR_Y,
216 |                                minPWM,
217 |                                maxPWM);
218 |     }
219 |
220 | > else { ...
223 | }
224 | }
```

Figure 30: Ré para esquerda

- **Ré** - Se o analógico direito for pressionado para trás (-) e o analógico esquerdo 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;



```
187     else if (valorAnalogicoDireito < (centerAnalogR_Y - toleranciaAnalogico)) {
188
189         pwmMotorDireito1 = minPWM;
190         pwmMotorEsquerdo1 = minPWM;
191
192 >     if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) { ...
204     }
205
206 >     else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) { ...
218     }
219
220     else {
221         pwmMotorDireito2 = map(valorAnalogicoDireito, centerAnalogR_Y, minAnalogR_Y, minPWM, maxPWM);
222         pwmMotorEsquerdo2 = map(valorAnalogicoDireito, centerAnalogR_Y, minAnalogR_Y, minPWM, maxPWM);
223     }
224 }
```

Figure 31: Ré

- **Rotação horária no próprio eixo** - Se o analógico esquerdo for pressionado para a direita (+) e o analógico direito não for pressionado (0), o robô deve rotacionar no sentido horário. Para esse 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 (robô rotaciona no próprio eixo);
- **Rotação anti-horária no próprio eixo** - Se o analógico esquerdo for pressionado para a esquerda (-) e o analógico direito não for pressionado (nulo), o robô deve rotacionar no sentido anti-horário. Para esse 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 (robô rotaciona no próprio eixo);
- **Imobilidade** - Se nenhum analógico for pressionado, o robô permanece imóvel.

```
226     else {
227
228         if (valorAnalogicoEsquerdo > (centerAnalogL_X + toleranciaAnalogico)) {
229             pwmMotorDireito1 = minPWM;
230             pwmMotorEsquerdo1 = maxPWM;
231             pwmMotorDireito2 = map(valorAnalogicoEsquerdo, centerAnalogL_X, maxAnalogL_X, minPWM, maxPWM);
232             pwmMotorEsquerdo2 = map(valorAnalogicoEsquerdo, centerAnalogL_X, maxAnalogL_X, maxPWM, minPWM);
233         }
234
235         else if (valorAnalogicoEsquerdo < (centerAnalogL_X - toleranciaAnalogico)) {
236             pwmMotorDireito1 = maxPWM;
237             pwmMotorEsquerdo1 = minPWM;
238             pwmMotorDireito2 = map(valorAnalogicoEsquerdo, centerAnalogL_X, minAnalogL_X, maxPWM, minPWM);
239             pwmMotorEsquerdo2 = map(valorAnalogicoEsquerdo, centerAnalogL_X, minAnalogL_X, minPWM, maxPWM);
240         }
241
242         else {
243             pwmMotorDireito1 = minPWM;
244             pwmMotorEsquerdo1 = minPWM;
245             pwmMotorDireito2 = minPWM;
246             pwmMotorEsquerdo2 = minPWM;
247         }
248     }
```

Figure 32: Rotação no próprio eixo e imobilidade

Foi definida uma **zona morta para o controle**, ou seja, se os analógicos 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.



As variáveis `centerAnalogR_Y` e `centerAnalogL_X` representam, respectivamente, o valor que o analógico direito e esquerdo retornam quando não estão sendo pressionados.

As variáveis `minAnalogR_Y` e `maxAnalogR_Y` representam, respectivamente, o menor e o maior valor que o analógico direito pode retornar quando verticalmente movimentado (valor quando o joystick é inteiramente pressionado trás e para a frente).

As variáveis `minAnalogL_X` e `maxAnalogL_X` representam, respectivamente, o menor e o maior valor que o analógico esquerdo pode retornar quando horizontalmente movimentado (valor quando o joystick é inteiramente pressionado esquerda e para a direita).

Esses seis valores variam (dentro da faixa informada no início desse tópico) de controle para controle e podem ser obtidos utilizando um programa que será demonstrado mais a diante. É importante lembrar que, **no código principal, o valor do analógico direito é multiplicado por -1 ao ser lido, portanto, deve-se fazer essa consideração na hora de definir os limites do controle** (no arquivo `parâmetros.h`).

As variáveis `minPWM` e `maxPWM` armazenam os limites da faixa PWM para os quais as combinações dos analógicos devem ser convertidas. Conforme visto na apostila sobre Hardware, a saída analógica da ESP32 é capaz de gerar valores que vão de 0 a 255. Porém, a combinação dos valores dos analógicos 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.

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

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 analógicos (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 a diferença entre o PWM de cada pino, maior a rotação do motor.

DICA: no código disponibilizado, a variável `maxPWM` está definida para 200. **É possível aumentar a rotação dos motores alterando a variável para 255.** Entretanto, é importante mencionar que essa alteração fará com que os motores esquentem um pouco mais, pois trabalharão com uma tensão superior a sua tensão nominal (overvoltage), o que pode **danificar o componente**.

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 do motor forem corretamente conectados à placa eletrônica. Para resolver esse problema, projetou-se um trecho código que permite a inversão do sentido de cada motor, de forma individual. **A inversão do motor direito é feita pelas setas CIMA e BAXIO e a do motor esquerdo, pelas setas DIREITA e ESQUERDA.**

Além dos nomes atribuídos aos pinos de controle dos motores de locomoção no arquivo `parâmetros.h`, há também um nome "lógico" para esses pinos, permitindo que a inversão aconteça. Ao pressionar uma



das setas, o nome lógico de cada pino é alterado, invertendo o valor PWM recebido por cada pino e, consequentemente, invertendo a rotação do motor.

```
250 | | | /* ----- Lógica de inversão de giro da movimentação ----- */
251 |
252 | uint8_t leituraSetinhas = myController->dpad();
253 |
254 | // Se seta cima ou seta baixo forem pressionadas, inverte sentido motor direito
255 | if (leituraSetinhas == 0x01)
256 | | sentidoMotorDireito = PINO_1_MOTOR_DIREITO, velocidadeMotorDireito = PINO_2_MOTOR_DIREITO;
257 | else if (leituraSetinhas == 0x02)
258 | | sentidoMotorDireito = PINO_2_MOTOR_DIREITO, velocidadeMotorDireito = PINO_1_MOTOR_DIREITO;
259 |
260 | // Se seta esquerda ou seta direita forem pressionadas, inverte sentido motor esquerdo
261 | if (leituraSetinhas == 0x04)
262 | | sentidoMotorEsquerdo = PINO_1_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_2_MOTOR_ESQUERDO;
263 | else if (leituraSetinhas == 0x08)
264 | | sentidoMotorEsquerdo = PINO_2_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_1_MOTOR_ESQUERDO;
```

Figure 33: Trecho para inversão de rotação dos motores

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 remove-se todos os dispositivos que haviam sido pareados anteriormente. Depois, define-se os pinos nos quais as pontes H estão conectados como saída (OUTPUT), utilizando o comando `pinMode`, e assume-se uma configuração de rotação inicial para os motores de locomoção. Por fim, garante-se que todos os motores estejam desligados (`desligaRobo()`).



```
287 void setup() {
288
289     Serial.begin(115200);
290
291     // Inicia comunicação Bluetooth que permite a conexão com controle.
292     BP32.setup(&onConnectedController, &onDisconnectedController);
293     BP32.enableVirtualDevice(false);
294
295     // Despara os controles que haviam sido conectados anteriormente.
296     BP32.forgetBluetoothKeys();
297
298     // Configura pinos da ESP32 para controle dos motores de arma.
299     pinMode(PINO_1_ARMA1, OUTPUT);
300     pinMode(PINO_2_ARMA1, OUTPUT);
301
302     pinMode(PINO_1_ARMA2, OUTPUT);
303     pinMode(PINO_2_ARMA2, OUTPUT);
304
305     // Configura pinos da ESP32 para controle dos motores de locomoção.
306     pinMode(PINO_1_MOTOR_ESQUERDO, OUTPUT);
307     pinMode(PINO_2_MOTOR_ESQUERDO, OUTPUT);
308
309     pinMode(PINO_1_MOTOR_DIREITO, OUTPUT);
310     pinMode(PINO_2_MOTOR_DIREITO, OUTPUT);
311
312     sentidoMotorEsquerdo = PINO_1_MOTOR_ESQUERDO, velocidadeMotorEsquerdo = PINO_2_MOTOR_ESQUERDO;
313     sentidoMotorDireito = PINO_1_MOTOR_DIREITO, velocidadeMotorDireito = PINO_2_MOTOR_DIREITO;
314
315     // Desliga movimentação e arma do robô.
316     desligaRobo();
317 }
```

Figure 34: 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 interrupta, 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.

```
319 void loop() {
320     // Checa se houve atualização nos dados do controle
321     bool dataUpdated = BP32.update();
322
323     // Se sim, chama a função processControllers() para processar os dados
324     if (dataUpdated)
325     |   processControllers();
326 }
```

Figure 35: 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.

Para conhecer um pouco mais sobre essas diretivas, consulte o link abaixo.

<https://learn.microsoft.com/pt-br/cpp/preprocessor/preprocessor-directives?view=msvc-170>

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>
```

Figure 36: Inicialização do arquivo .h

Logo abaixo vem a definição dos pinos utilizados para controlar os motores de movimentação e de arma. Digitar no código 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
```

Figure 37: Defines pinos motores



Comentando sobre os dados salvos no arquivo, primeiramente vem os parâmetros dos analógicos 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 válido lembrar que estes valores estão relacionados à resolução da saída analógica (PWM) da ESP32.

```
23 const int32_t minAnalogR_Y = -508, centerAnalogR_Y = 0, maxAnalogR_Y = 512; // Valores reais * -1
24 const int32_t minAnalogL_X = -512, centerAnalogL_X = 0, maxAnalogL_X = 508;
25
26 const int32_t toleranciaAnalogico = 10; // zona morta do controle
27
28 // Pino de controle em HIGH: 255 (menor velocidade) a 0 (maior velocidade)
29 // Pino de controle em LOW: 0 (menor velocidade) a 255 (maior velocidade)
30
31 const int minPWM = 0, maxPWM = 200; // valores limite para o PWM da ESP32
```

Figure 38: 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.