INSTITUTO FEDERAL DO ESPÍRITO SANTO

MESTRADO PROFISSIONAL EM

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Relatório de Robótica parte 1
Robô Omnidirecional

Demandante: Cassius Zanetti Resende

Coordenador/Pesquisador: Jorge Ruan Zeferino Conti

Pesquisador: Diego Santana

Serra 2018

Sumário

Objetivos 3
Materiais e métodos 4
Instalação da IDE do *arduino* e reconhecimento do *Esp32* na IDE 5
Pinagem *Esp32* 10
Sistema operacional de tempo real 11
Acionamento do motor de passo 12
Controle de baixo nível 13
Comunicação WIFI 14
Modelo do robô 15
Verificação de inicialização 16
Cronograma 17
Acompanhamento 18

Objetivos

Construir um robô omnidirecional com o auxílio de impressão 3D, partes estruturais e rodas omnidirecionais serão impressas em ABS.

Desenvolver um firmware responsável por realizar o controle de baixo nível.

Implementar a comunicação via *WIFI* entre o *Esp32* com o *MatLab*.

Desenvolver o controle de alto nível para o robô na plataforma *MatLab*.

Materiais e métodos

Para desenvolver o firmware será utilizado o *Esp32*, ilustrado na Figura 1, como *hardware* embarcado do robô, ele será responsável por controlar quatro motores de passo 28BYJ-48 com auxílio do driver ULN2003 ilustrados na Figura 2. O *Esp32* é responsável pelo controle de baixo nível dos quatro motores de passo, este *hardware* também vai estabelecer comunicação via *wifi* com um computador utilizando a plataforma *MatLab*, este computador será responsável por realizar o controle de alto nível e enviar para o robô a trajetória ponto a ponto desejada em intervalos de 100 milissegundos por ponto.

Fonte: https://2.bp.blogspot.com/-6b2Bg9WQgp0/WnHMKK5fqvI/AAAAAAAABZc/rLtEsp5RU8gAeH0_MmzCuBANI7iSKEoqgCLcBGAs/s1600/Pinagem.png

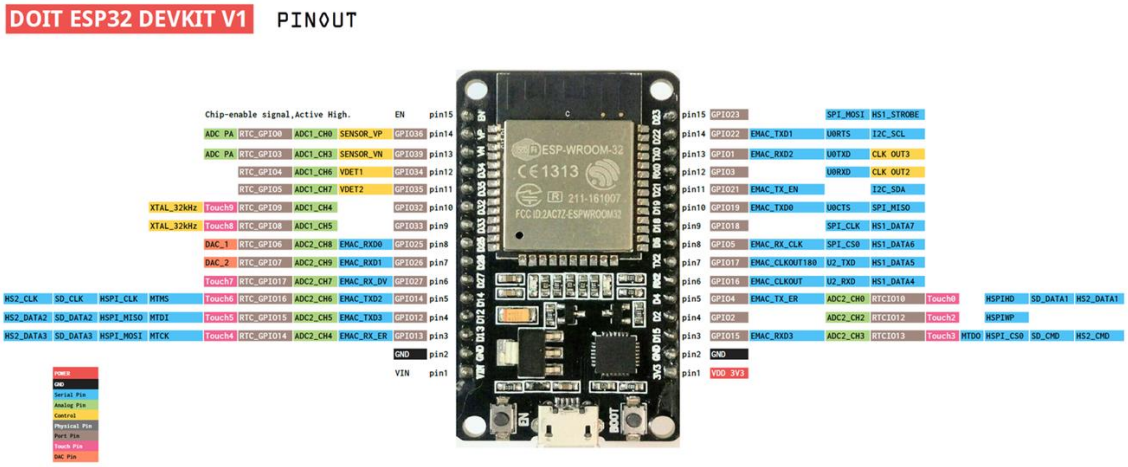


Figura 1 - Esp32

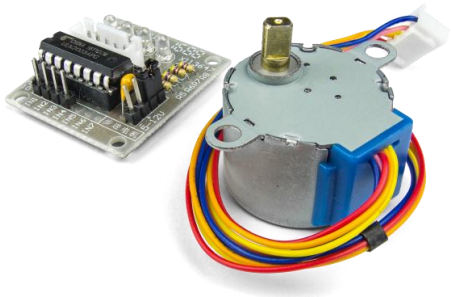


Figura 2 - Motor de passo 28BYJ-48 e Driver ULN2003

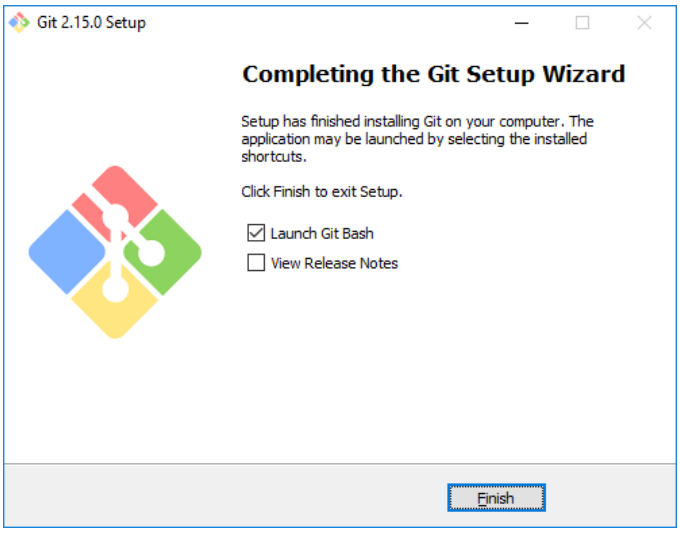
Instalação da IDE do *arduino* e reconhecimento do *Esp32* na IDE

O tutorial descrito a seguir de instalação, possui instruções que foram retiradas do link: https://www.arduinoecia.com.br/2017/11/modulo-esp32-wifi-bluetooth-ide-arduino.html

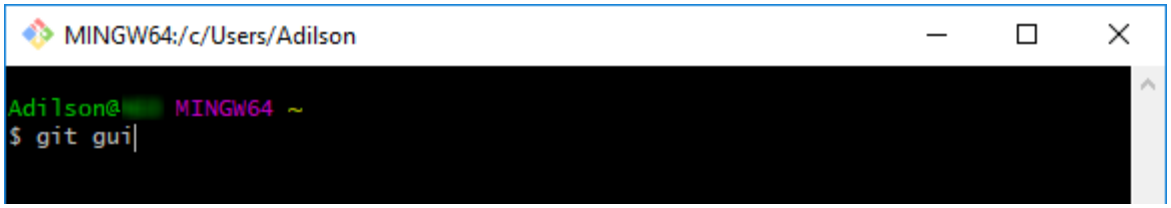
Lembrando este tutorial foi feito utilizando o Windows 10.

1 - Instale a **IDE do Arduino** (se você já não tem, pode encontrá-la no https://www.arduino.cc/en/Main/Software)

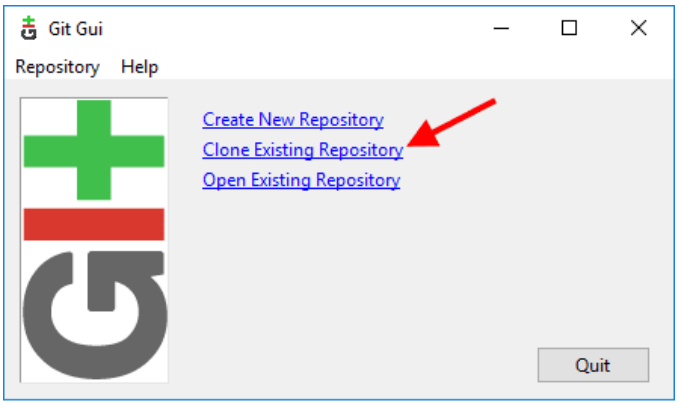
2 - Faça o download do **Git** https://git-scm.com e em seguida instale o programa (não foi alterado nenhuma opção durante a instalação, foi utilizado tudo no padrão sugerido pelo instalador). Ao final, selecione a opção **Launch Git Bash** e clique em Finish:



3 - Será então exibida a seguinte janela. Nela, digite o comando "**git gui**" e pressione ENTER:



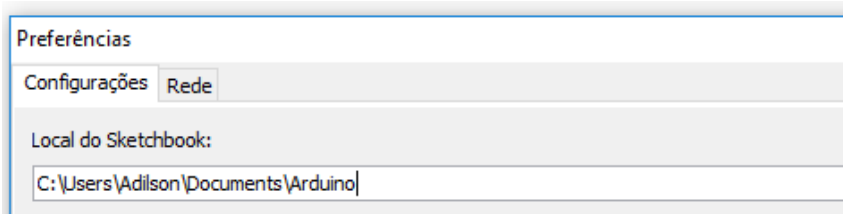
4 - Na janela Git GUI, selecione **Clone Existing Repository:**



5 - Na janela seguinte, preencha os campos **Source Location** e **Target Directory**:

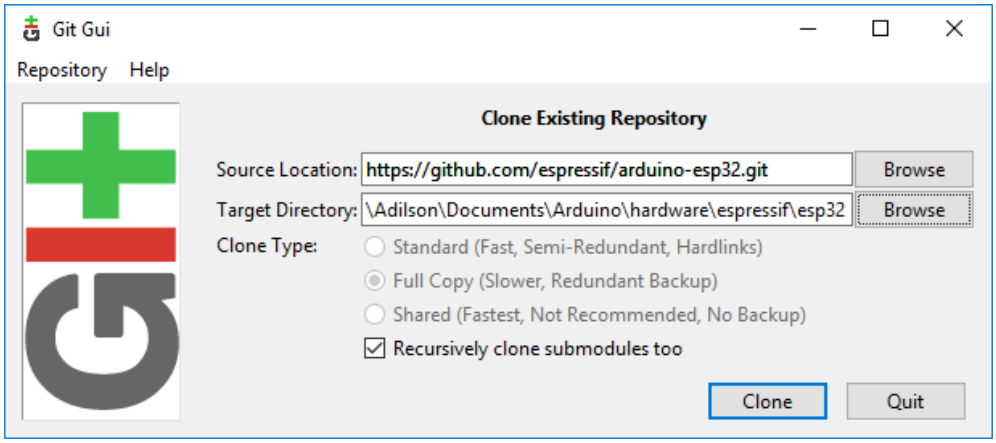
Source Location: https://github.com/espressif/arduino-esp32.git

O campo **Target Directory** pode variar, dependendo de como foi configurada a sua IDE, mas geralmente a pasta é *C:\users\[usuario]\Documents\Arduino*. Você pode encontrar essa informação dentro da IDE do Arduino, no menu **Arquivo => Preferências**:



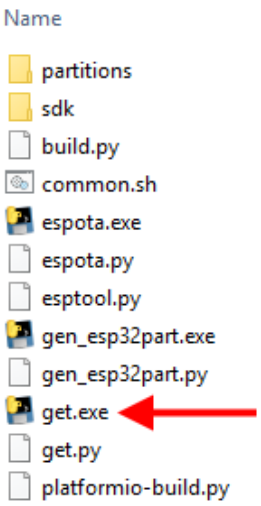
O campo Target Directory é formado pelo caminho mostrado em "Local do Sketchbook" mais *\hardware\espressif\esp32*. No meu caso então, o campo ficou assim:

Target Directory: C:\Users\Adilson\Documents\Arduino\hardware\espressif\esp32

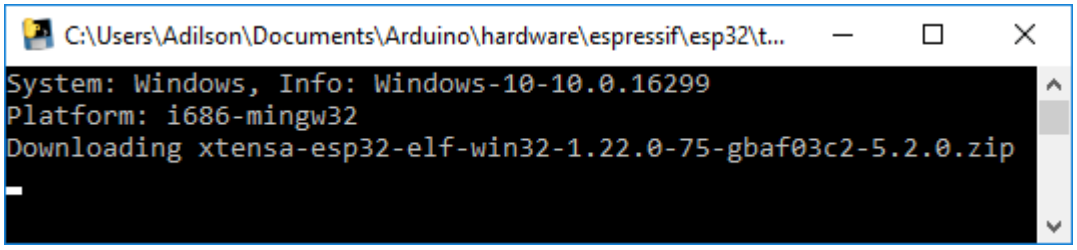


Após preencher os campos, clique em **Clone** e aguarde o download e instalação dos arquivos.

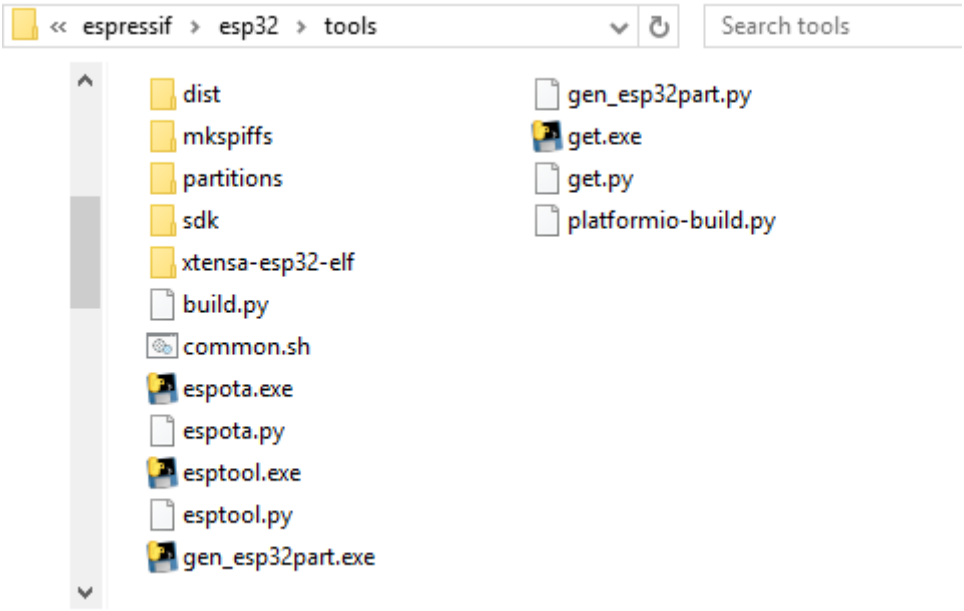
6 - Vá até a pasta **[Local do Sketchbook]\hardware\espressif\esp32\tools** e execute o arquivo **get.exe**:



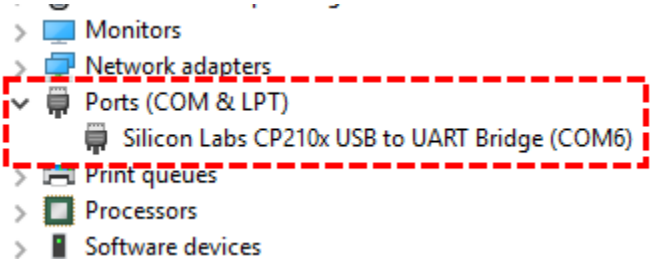
Aguarde o download e descompactação dos arquivos:



7 - Ao término do processo, você deve ter os arquivos abaixo na pasta:

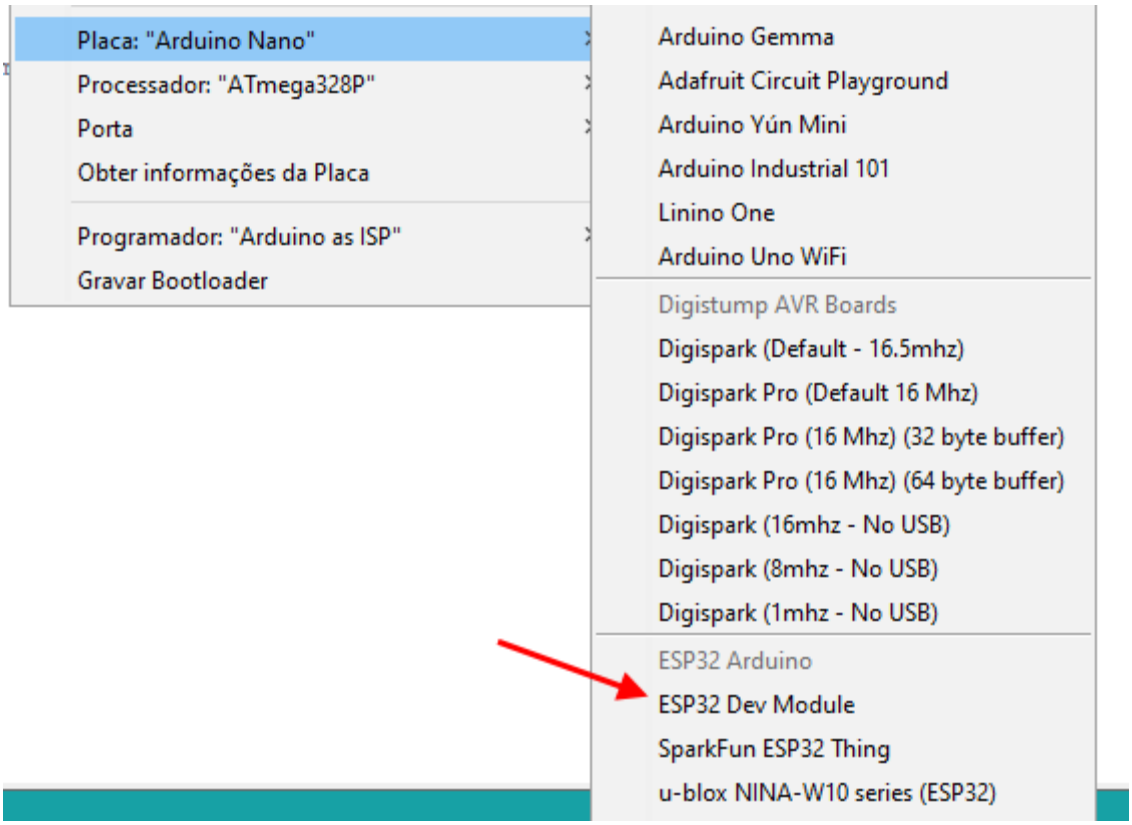


8 - Hora de conectar o módulo ESP32 na porta USB e aguardar a instalação dos drivers. No meu computador, pelo gerenciador de dispositivos vejo que foi adicionado um dispositivo chamado **Silicon Labs CP210x USB to UART Bridge** na porta **COM6**:



Se precisar dos drivers para o CP2102, recomendo este link.

9 - Inicie a IDE do Arduino e no menu Ferramentas => Placa selecione **ESP32 Dev Module**:

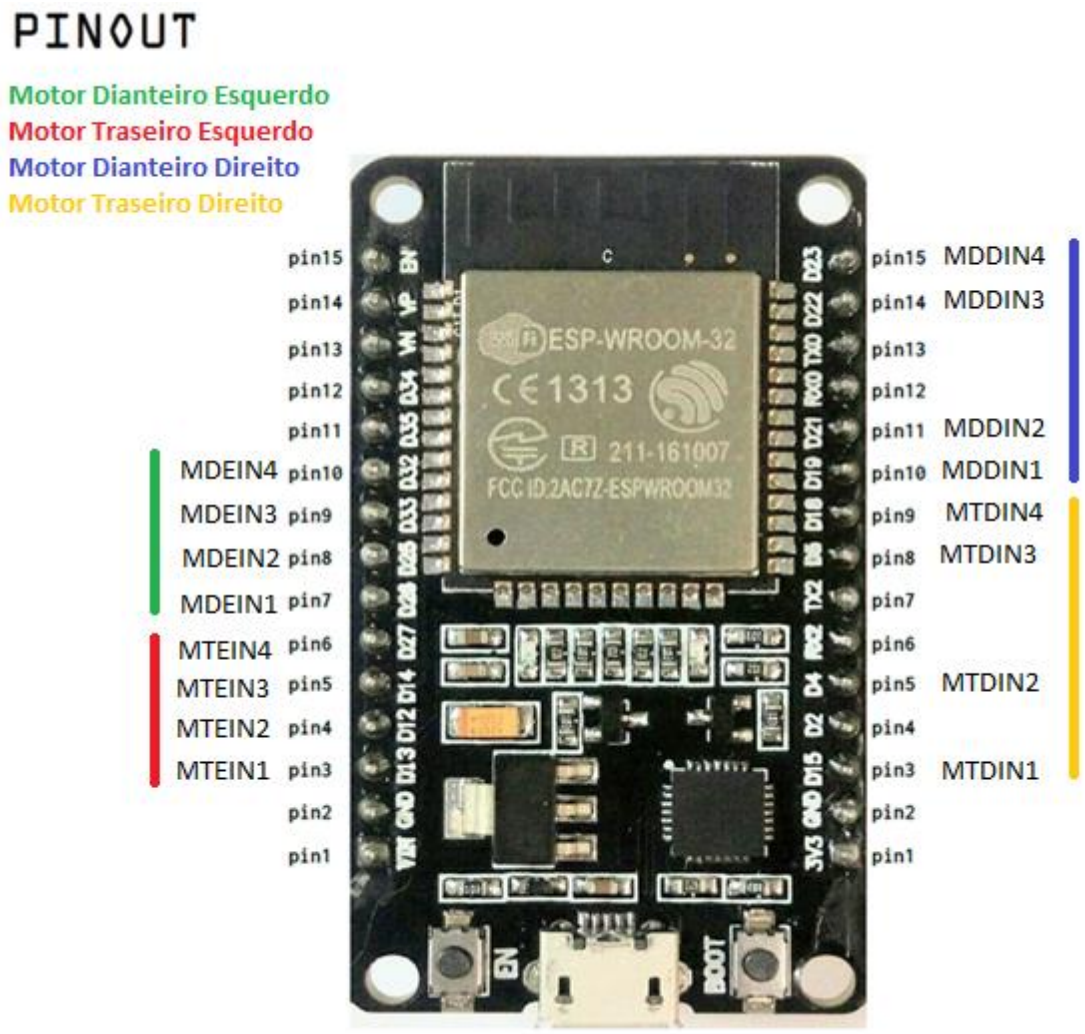


10 - Ainda no menu Ferramentas, selecione a porta COM do módulo. No meu caso, a porta **COM6**.

Agora o seu *Esp32* está pronto para receber firmware de programação.

Pinagem *Esp32*

A pinagem selecionada para conectar os drives ULN2003 corretamente seguindo a ordem de conexão dos fios assim como o posicionamento dos motores no robô. É muito importante a correta instalação dos drives e posicionamento dos motores para que o robô possa atuar como o esperado.



Sistema operacional de tempo real

Foi implementado o sistema operacional de tempo real ao *Esp32* para prover uma melhor qualidade de gerenciamento das tarefas desenvolvidas para o controle do robô omnidirecional, a qual é garantido a execução de determinada ação no intervalo de tempo definido. Para melhor entendimento das funcionalidades do processo de gerenciamento de tarefas em tempo real do *FreeRTOS,* é aconselhado realizar a leitura no material de apoio ao qual a própria ferramenta disponibiliza no link https://www.freertos.org/implementation/a00002.html

Funções utilizadas do *FreeRTOS*:

```
xTaskCreate(taskStepper1, "taskStepper1", 5000, NULL, 100, &xHandleStepper1);
xTaskGetTickCount();
vTaskDelay( xDelay );
vTaskDelayUntil( &xLastWakeTime, xDelay );
vTaskResume(xHandleStepper1);
vTaskSuspend(xHandleStepper1 );
vTaskDelete( NULL );
```

Acionamento do motor de passo

O acionamento unipolar do motor foi programado utilizando passo completo seguindo a Tabela 1 para a sequência de acionamento das bobinas do motor. É possível acionar o motor apenas com uma bobina por vez como demonstra na Tabela 2.

Tabela 1 – Fullstep duas bobinas

Passo	1a	2a	1b	2b
1	ativo	desativado	desativado	ativo
2	ativo	ativo	desativado	desativado
3	desativado	ativo	ativo	desativado
4	desativado	desativado	ativo	ativo

Tabela 2 - Fullstep apenas uma bobina

Passo	1a	2a	1b	2b
1	ativo	desativado	desativado	desativado
2	desativado	ativo	desativado	desativado
3	desativado	desativado	ativo	desativado
4	desativado	desativado	desativado	ativo

Este acionamento foi incorporado a biblioteca *StepperControl.cpp* pela função *stepMotor*.

A seleção de qual tabela será utilizada para o acionamento dos motores é necessário enviar um caractere de configuração, caso seja enviado ‘B’ a primeira tabela é utilizada caso envie ‘b’ a segunda tabela é empregada, por default o ‘b’ é iniciado.

Controle de baixo nível

O código desenvolvido para manipulação de baixo nível do motor pode ser observado na imagem a seguir, este código é similar para todos os motores.

Esta tarefa possui um loop infinito no qual tem intervalos de atualização controlados pelo *xDelay* o qual é modificado com auxílio do Hz1, que por sua vez é calculado a partir de informações que chegam via *wifi* contendo dados da trajetória a qual o robô deverá percorrer.

```
//////////////////////////////////////////taskStepper1/////////////////////////////////////////////////////


void taskStepper1( void * parameter)
{
  int hertz = Hz1; // entrada de velocidade do motor em hertz
  int sentido = 1;
  float amostragem = 1000.0 / (abs(hertz) * 4.0); //conversão de hertz em amostragem
  TickType_t xDelay = amostragem / portTICK_PERIOD_MS;
  int cont = 1, limitMax = 0;

  for (;;) {
    hertz = Hz1; // velocidade em hertz desejada para o motor

    if (hertz != 0) {// proteção contra velocidades nulas

      if (hertz > 120)  //define limites de velocidade superior
        hertz = 120;

      if (hertz < -120) //define limites de velocidade inferior
        hertz = -120;

      if (hertz > 0) { //define o sentido de giro e salva a odometria
        sentido = 1;
        odometrial++;
      }
      else {
        sentido = -1;
        odometrial--;
      }

      Stepper1.step(sentido); // alterna o chaveamento do bobinamento

      amostragem = 1000.0 / (abs(hertz) * 4.0); // encontra o valor em milissegundos que deve ser aplicado

      limitMax = floor(10 * (amostragem - floor(amostragem))); // pega o valor da primeira casa decimal

      if (cont <= limitMax) // define os tempos de comutação do motor
        xDelay = floor(amostragem) + 1;//delay extra pra compensar o arredondamento
      else
        xDelay = floor(amostragem);

      cont++;
      if (cont > 10) // controla o limite do contador, máximo 10 interações
        cont = 1;

      vTaskDelay( xDelay ); // aplica o delay da tarefa
    }
    else {// protege o RTOS quando velocidade do motor for zero, amostragem tende a infinito
      Stepper1.stopMotor(); // desenergiza o motor
      vTaskSuspend( xHandleStepper1 );
    }
  }
  Stepper1.stopMotor(); // desenergiza o motor
  Serial.println("Motor 1 desligado"); // avisa pelo seria q a tarefa foi encerrada
  vTaskDelete( NULL );//deleta a tarefa e libera o espaço utilizado
}
```

Comunicação WIFI

A comunicação via *WIFI* foi implementada ao Esp32 com o auxílio da biblioteca <WiFi.h> a qual possibilita criar no robô um *acess point*.

Como o intuito deste trabalho é abordar apenas o controle de um robô utilizando o *wifi* como meio de comunicação com um notebook, comunicação ponto a ponto, adotou-se este método no qual o robô possibilita a geração de ponto de acesso removendo a necessidade de investimento em equipamentos de rede como roteador.

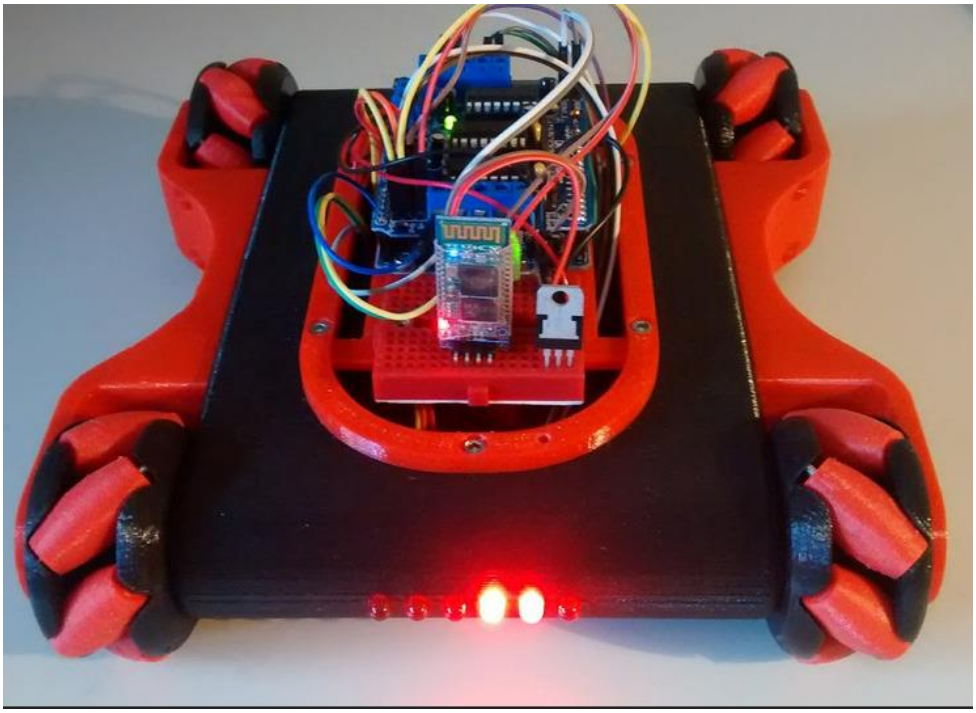
Com isso ao ligar o robô omnidirecional automaticamente é inicializado uma rede criada por ele especificamente para receber e enviar dados pertinentes ao controle do mesmo.

No *MatLab* foi utilizado a função “tcpip” configurada para se conectar na rede gerada pelo robô, desta maneira após a inicialização do robô é necessário fazer *login* pelo notebook na rede do robô, assim possibilitando a conexão da função “tcpip” na rede do robô. E posterior envios de comandos pela rede.

Modelo do robô

O modelo de robô que busca ser impresso em 3D é similar ao da imagem abaixo.

Por conta da demanda da impressora em outros projetos e por períodos de manutenção ainda não foi possível realizar a impressão de todas as partes do robô, sendo assim o cronograma inicial que era previsto para entrega da “Construção da estrutura do carro” até o primeiro relatório terá seu prazo estendido para o segundo relatório.



Verificação de inicialização

Ao inicializar o robô omnidirecional ele realizara uma a energização individual e progressiva de cada bobina seguindo uma ordem para os motores, como o driver possui *LEDS* indicadores é importante observar durante essa inicialização se todos os *LEDS* serão acesos, isso garantira que todos os pinos estão conectados permitindo o perfeito funcionamento dos motores.

Após esse processo um *LED* azul no Esp32 piscara a cada cinco segundos indicando que o robô está em operação pronto para realizar a conexão via *WIFI*, após o notebook acessar a rede criada pelo Esp32 e uma conexão seja estabelecida via *software* o *LED* azul passara a piscar duas vezes a cada dois segundos, indicando que o robô possui uma conexão via *WIFI* consolidada com o notebook.

Cronograma

Atividade	Responsável	15/10	15/11	15/12
Construção da estrutura do carro	Diego	X	X	
Construção da Roda omnidirecional	Diego	X	X	
Controle Baixo Nível	Jorge	X		
Conexão de comunicação *wifi Esp32* com *MatLab*	Jorge	X		
Comando de Alto nível/odométrica	Jorge	X	X	
Dimensionamento de bateria	Diego	X	X	
Montagem	Diego/Jorge		X	X
Teste/controle	Diego/Jorge			X
Entrega de relatórios parciais	Jorge	X	X	X

Acompanhamento

Seguindo o cronograma foi desenvolvido o firmware responsável pelo controle de baixo nível dos motores, assim como a comunicação via *WIFI* disponível em hardware do próprio *Esp32*.

Todo o *firmware* desenvolvido foi implementado ao *Esp32* seguindo a estrutura de sistema operacional de tempo real disponibilizada pela *FreeRTOS*, o qual garante a execução das tarefas que lhe foram implementadas em intervalo de tempo determinado, gerando maior confiabilidade para o *firmware*.

A impressão das rodas e da parte estrutural do carro encontrasse em processo, visto que a impressora 3D possui outras demandas de projetos e por conta de alguns imprevistos com manutenções o tempo de espera foi afetado, assim o que era previsto no cronograma teve de ser alterado, agora o prazo da entrega da parte estrutural do carro ficou para a segunda parte do relatório.