UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
Bacharelado em Sistemas de Informação
Wallace Cristian da Silva

UMA APLICAÇÃO ROBÓTICA INTEGRANDO O ROS E ARDUINO

Diamantina
2019

Wallace Cristian da Silva

UMA APLICAÇÃO ROBÓTICA INTEGRANDO O ROS E ARDUINO

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação, como parte dos requisitos exigidos para a obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Rafael Santim

Diamantina
2019

Wallace Cristian da Silva
UMA APLICAÇÃO ROBÓTICA INTEGRANDO O ROS E ARDUINO / Wallace Cristian da Silva. – Diamantina, 2019-
51 p. : il. 30 cm.

Orientador: Rafael Santim

Trabalho de Conclusão de Curso –
, 2019.

1. ROS. 2. Arduino. 2. NodeMCU ESP8266-12 V2. I. Prof. Me. Rafael Santim. II. Universidade Federal dos Vales do Jequitinhonha e Mucuri. III. Faculdade de Ciências Exatas e da Terra. IV. Uma Aplicação Robótica Integrando o ROS e Arduino.

UMA APLICAÇÃO ROBÓTICA INTEGRANDO O ROS E ARDUINO

Wallace Cristian da Silva

Orientador Profº Me. Rafael Santin

Monografia submetida à Banca Examinadora designada pelo curso de Sistemas de Informação da Universidade Federal dos Vales do Jequitinhonha e Mucuri, como requisito para obtenção do título de Bacharel em Sistemas de Informação.

APROVADO em 24 / 01 /2019

Marcelo Ferreira Rego

Profº Me. Marcelo Ferreira Rego - UFVJM

Alan Fernando Santos de Ávila

Analista de Tecnologia da Informação Me. Alan Fernando Santo de Ávila – UFVJM / Departamento da Computação

Rafael Santin

Profº Me Rafael Santin. – UFVJM (Orientador)

A minha mãe, Maria Nazaré da Cruz Silva!

AGRADECIMENTOS

Primeiramente, agradeço a N. Sra. Aparecida, a quem me apeguei nos momentos desesperadores e que nunca me abandonou.

À minha mulher e filha,que sempre me apoiaram.

Quero agradecer ao meus colegas de curso que sempre estiveram ao meu lado nos momentos difíceis me ajudando a vencer as dificuldades.

Não poderia deixar de agradecer ao meu orientador Prof. Dr.Rafael Santin, o qual me apoiou nos momentos mais conturbados do projeto ajudando no desenvolvimento e orientado quanto às soluções.

Aos meus amigos de serviço da 36 CIA/3BPM que sempre estiveram ao meu lado, que sempre me amparam com trocas de serviço para que pudesse assistir às aulas, em especial ao escalante que sempre fez o possível para me ajudar.

E a todos que contribuíram para que este momento fosse possível!

”Se quer viver uma vida feliz, amarre-se a uma meta, não às pessoas nem às coisas”.

Albert Einstein

RESUMO

A monografia apresenta o ROS(Sistema Operacional Robótico) um software com funcionalidades e serviços para aplicações robóticas integrado ao Arduino uma plataforma barata , funcional, de fácil programação e de valor econômico acessível. Neste contexto, foi montado um robô aproveitando as ferramentas e bibliotecas do ROS e as facilidades de prototipagem do Arduíno na montagem dos circuitos eletrônicos, além do uso de suas bibliotecas para sensores e atuadores. O projeto promoveu a comunicação entre o computador e o robô, através da rede Wifi local por meio de mensagens do ROS, para atribuir ao robô uma nova rota após a detecção de colisão.

Palavras-chave: ROS. Arduino. NodeMCU ESP8266-12 V2 .

ABSTRACT

The monograph presents the ROS (Robotic Operating System), a software with functionalities and applications for integrated robotic applications, an inexpensive platform, functional, easy to program and affordable economic value. This content was assembled in conjunction with ROS tools and libraries and Arduino prototyping facilities in the assembly of electronic circuits, as well as being used for the selection of sensors and actuators. The project promotes communication between the computer and the robot through the local Wifi network through ROS messages to assign a rapid change to collision detection.

Keywords: ROS. Arduino. NodeMCU ESP8266-12 V2

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ROS - Sistema Operacional Robótico

SO - Sistema Operacional

UFVJM - Universidade Federal dos Vales do Jequitinhonha e Mucuri

BSD - Berkeley Software Distribution

IDE - Integrated Development Environment

MQTT -Message Queuing Telemetry Transport

TCP - Transmission Control Protocol

IP - Internet Protocol address

I2C - Inter-Integrated Circuit

SPI -Serial Peripheral Interface

USB - Universal Serial Bus

SUMÁRIO

1 INTRODUÇÃO

Esta monografia teve por objetivo desenvolver uma aplicação robótica utilizando o Sistema Operacional Robótico (ROS), em comunicação com a plataforma Arduino, que possui características tais como: ser funcional, de fácil programação e de valor econômico acessível.

Para o desenvolvimento do trabalho, foi realizada a montagem de um robô utilizando as tecnologias: do Arduino, com suas facilidades de prototipagem e as bibliotecas prontas para sensores e atuadores;e do ROS, com as abstrações de *hardware* e a comunicação por mensagens.

1.1 Problema

A interação entre o Arduino e o ROS com o Rosserial Arduino, só é possível utilizando um cabo USB conectado ao computador e ao robô, ou através do uso de um mini-computador sobre o chassi robótico onde seria instalado o Ubuntu e posteriormente o ROS, que estaria conectado ao Arduino através do cabo USB.

Contudo, o objetivo do trabalho era fazer uma conexão em rede local sem fios, diante disso, buscamos a solução na comunidade Iot(Internet das Coisas), que utiliza o *hardware* NodeMCU ESP8266-12 V2 para conexão de dispositivos a rede de internet, o qual, permite o uso da biblioteca Rosserial Arduino e permite a conexão sem Wi-fi.

O NodeMCU ESP8266-12 V2 agrega a plataforma de desenvolvimento da linguagem de programação Lua, possui conversor USB serial e regulador de tensão, sua pinagem facilita a montagem de pequenos circuitos e assim como o Arduino é de fácil programação e de baixo custo(OLIVEIRA, 2017).

1.2 Justificativa

A simplicidade do Arduino permite ao usuário a criação de diversos projetos sem que ele tenha um conhecimento aprofundado de eletrônica e programação, com isso, leigos, autodidatas e acadêmicos podem criar projetos que atenderão suas necessidades(THOMSEN, 2014).

Contudo, observamos um aumento na complexidade desses projetos, pois, a cada ideia observa-se um número crescente de sensores e atuadores conectados aos dispositivos embarcados, que em sua maioria, não possue capacidade de processamento suficiente para tratar os dados disponibilizados por estes dispositivos e disparar as ações programadas pelos desenvolvedores.

O crescimento da complexidade dos projetos que envolvem o Arduino está ligada principalmente ao baixo custo do *hardware*, facilidade de pagamento e as lojas *online*, as quais, além de oferecer o produto, trazem em suas páginas a documentação, além de tutoriais em vídeo do uso dos dispositivos.

Diante desse cenário, surge o ROS (*Robot Operating System*), um *framework* que traz em seu pacote bibliotecas e ferramentas prontas para a implementação de sistemas robóticos.

O ROS, ainda traz em seu portfólio um sistema de comunicação por mensagens em rede, isso possibilita que sensores enviem mensagens a quem interessar na rede, essas são recebidas por pontos na rede conhecidos como nós, que disparão ações para outros nós, onde podem estar: motores, servos, *leds* e outros nós.

1.3 Objetivos

1.3.1 Objetivo Geral

Com foco na simplicidade oferecida pelo Arduino e pelo ROS, o objetivo geral é estabelecer a comunicação entre nós do ROS executando em diferentes máquinas por meio da rede *wi-fi* para controlar uma plataforma robótica de baixo custos com Arduino.

1.3.2 Objetivo Específico

- Criar um robô utilizando as ferramentas do Arduino.
- Entender como o ROS é utilizado no Arduino .
- Promover a publicação de mensagens do Arduino para o ROS.
- Promover a subscrição de mensagens do ROS para o Arduino, disparando ações no robô.

1.4 Visão geral do trabalho

A **Figura 1**, demonstra um diagrama geral do robô, que possui seis placas com as seguintes funções:

1. o notebook, onde está executando o ROS Master e o nó Controle;
2. o roteador, responsável pela rede Wi-fi;
3. o NodeMcu, é uma placa que realiza transmissão *Wi-fi* e a detecção de colisão através do sensor ultrassônico;
4. Conversor de Tensão, tem a função de integrar placas com tensões diferentes;
5. o sensor Ultrassônico, identifica os obstáculos e os transmite para o Arduino e o NodeMcu ao mesmo tempo;
6. Arduino Uno, responsável pelo controle de motores e detecção de colisão através do sensor ultrassônico;
7. o *Motor Driver* 2x2A, controla os motores a partir dos comandos do Arduino Uno;

8. Plataforma Robótica Falco, aloja todas as placas.

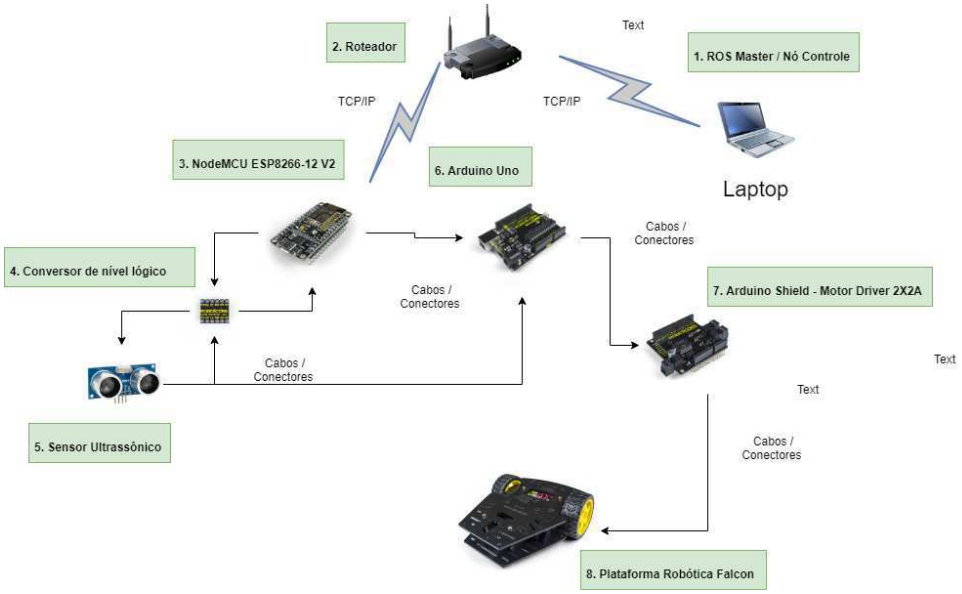

Figura 1 – Diagrama de montagem do robô.

Fonte:(Autor, 2019).

Diante da FIGURA 1, observamos o funcionamento do robô: através do sensor ultrassônico tanto o Arduino como o NodeMcu podem detectar a presença de um objeto.

Captado um obstáculo: o NodeMcu publica uma mensagem para o nó Controle, que está sendo executado no notebook; o Arduino Uno para o robô, até receber a nova rota; o nó controle publica a nova rota que é subscrita pelo NodeMcu; com a nova rota, o NodeMcu a transmite para o Arduino Uno, que utiliza o *motor driver* 2x2A para dar uma nova direção para o robô.

2 REFERENCIAL TEÓRICO

2.1 Sistema Operacional Robótico-ROS

O ROS (Sistema Operacional Robótico) faz parte de uma classe de *middleware* para robôs (CERIANI; MIGLIAVACCA, 2012). É um *software* livre, que foi desenvolvido por Morgan Quigley, na Universidade de Standford na década de 2000, sendo aperfeiçoado em 2008 por Willow Garage, é licenciado pela BSD, o que permite seu uso em projetos comerciais e não comerciais, sendo muito utilizado no meio acadêmico (LAGES, 2016).

O ROS viabiliza o desenvolvimento de sistemas robóticos, facilitando a interação entre diversos sensores e atuadores através de suas bibliotecas e ferramentas (CERIANI; MIGLIAVACCA, 2012). Esse sistema permite que cálculos ou funcionalidades mais complexas sejam distribuídos entre diversos processos ou nós em execução comunicando-se por meio de mensagens através do TCPROS, que utiliza o *socket* TCP/IP (UNKNOWNENTITY1, 2013).

Além disso, o ROS traz maior rapidez no desenvolvimento dos projetos devido a sua simplicidade, abstração de *hardware* e modularização no desenvolvimento dos processos (JUNIOR, 2016).

O sistema possibilita a implementação em múltiplas linguagens, tais como: c, c++, *Phyton*(JUNIOR, 2016), e disponibiliza bibliotecas que simplificarão a comunicação do ROS com diferentes aplicações(ROMAINREIGNIER, 2018) .

2.1.1 Fundamentos do ROS

A seguir são apresentados os principais conceitos do ROS, divididos basicamente em três níveis:

- Sistema de Arquivo,
- Grafo Computacional,
- Comunidade.

2.1.2 Sistema de Arquivo

O sistema de arquivos, tem como objetivo organizar a idealização do projeto em diretórios a exemplo da **Figura 2**, cada diretório agrupa diferentes informações referentes à suas funcionalidades, permitindo modificações (MAHTANI *et al.*, 2016).

Figura 2 – Espaço de trabalho ou *WorkSpace*

Fonte: (PACKT, 2016)

2.1.3 Grafo de computação

A interação entre os processos no ROS é representada por um grafo de computação, neste grafo todos os processos estão conectados, permitindo que nós interajam com outros nós. Além disso, é possível visualizar as informações que são transmitidas entres os processos (MAHTANI *et al.*, 2016).

As principais questões associadas ao grafo são os nós, *master*, mensagens e tópicos. Os conceitos referentes cada um desses itens são apresentados a seguir.

2.1.3.1 Nós

Os nós são processos em execução, podem ser em *roscpp* ou *rospy*, este arranjo possibilita que as funções do robô sejam divididas em módulos com funcionalidades específicas(MAHTANI *et al.*, 2016).

Conforme(SAITO, 2013), os comandos que são usualmente utilizados para manipular os nós são:

- **rosnode info [nome do nó]** : mostra informações sobre o nó;
- r**rosnode kill [nome do nó]** : termina a execução do nó;
- r**rosnode list :** lista todos os nós do *Master*;
- **rosnode *machine* ou *hostnome*:** lista os nós que estão sendo executados em uma máquina específica ou lista as máquinas;
- **rosnode ping [nome do nó]:** realiza o teste de conectividade dos nós;
- **rosnode cleanup :** elimina informações de registro de nós inacessíveis.

2.1.3.2 *Master*

O *Master* fornece informações para que os nós troquem dados, para isso, realiza o registro de serviços e nomes para cada nó. Seu objetivo é permitir a localização entre os nós, fazendo com que interajam através da comunicação par-a-par(MAHTANI *et al.*, 2016).

A **Figura 3** possui três nós: o primeiro é o *Master*, o segundo nó contém um sensor e faz à publicação dos dados deste sensor e o terceiro nó *rgt_polt* recebe os dados do sensor ou os subscrevem, todavia, tanto o nó que publica quanto o nó que subscreve têm que se registrar no *Master* para que possam realizar algum tipo de comunicação.

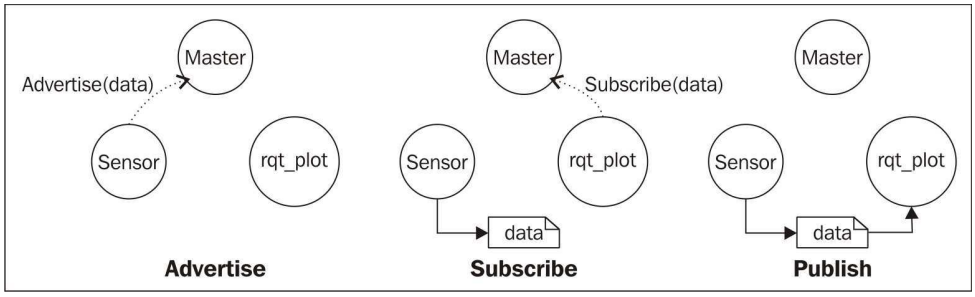

Figura 3 – Interação dos nós com o Master

Fonte:(PACKT, 2016)

2.1.3.3 Mensagens

Os dados são transmitidos entre os nós através de mensagens, que podem ser tipos primitivos de dados, como matrizes(MAHTANI *et al.*, 2016).

As mensagens ficam armazenadas em um subdiretório no pacote do ROS conforme **Figura 4**. (INSTITUTE, 2016).

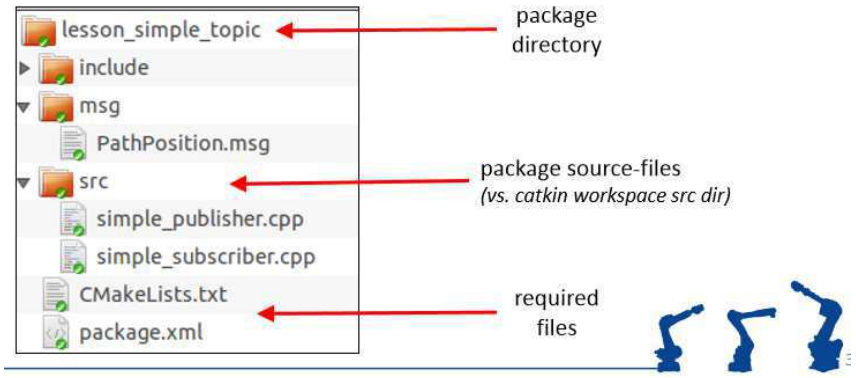

Figura 4 – Localização do diretório de mensagens

Fonte: (INSTITUTE, 2016)

Segundo (DIRKTHOMAS, 2017), os principais pacotes de mensagens são:

- **std_msg:** são mensagens com tipos primitivos de dados, como: *int , float, String , time*;
- ***geometry_*** **msgs:** Oferece mensagens primitivas comuns em forma de pontos, vértices e poses facilitando a comunicação entre sistemas que podem ou não ser semelhantes;
- **sensor_ msgs:** são mensagens utilizadas para sensores como: câmeras, sensores de *laser*, ultrassônicos, entre outros.

2.1.3.4 Tópicos

São barramentos utilizados pelos nós para publicar e sobrescrever mensagens, podendo ter vários assinantes e editores (MAHTANI *et al.*, 2016).

Na imagem **Figura 5**, visualizamos a publicação de uma mensagem do tipo *String*, sendo que temos dois nós que subscrevem esta mensagem.

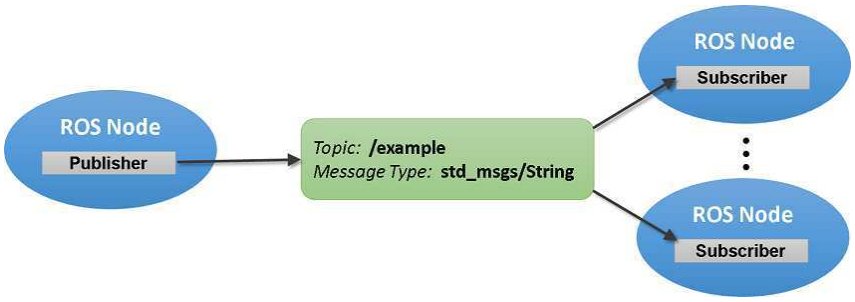

Figura 5 – Tópicos

Fonte: (MATHWORKS, 2018)

Os comandos comumente utilizados para a interação com os tópicos são, segundo (KENTAROWADA,):

- **rostopic bw /topic:** largura de banda usada no tópico;
- **rostopic echo / topic:** imprime as mensagens do ROS no terminal;
- **rostopic find message_type :** localiza o tópico pelo tipo de mensagem;
- **rostopic hz /topic:** exibe a taxa de publicação do tópico;
- **rostopic info /topic:** exibe informações do tópico escolhido;
- **rostopic list:** lista os tópicos ativos.

2.1.3.5 Serviços

Os serviços são uma comunicação sincronizada onde o nó servidor só envia resposta quando há uma solicitação do nó cliente (MAHTANI *et al.*, 2016).

2.1.4 Comunidade

É uma maneira do ROS permitir que comunidades desagregadas compartilhem conhecimento e software através de: distribuições, repositórios e ROS Wiki(MAHTANI *et al.*, 2016).

2.1.5 Rosserial Arduino

É um pacote que auxilia na comunicação serial entre os ROS e o Arduino, na **Figura 6** temos esta representação. O Arduino Uno pública as leituras do sensor para o notebook pela conexão serial(HENDRIX, 2014).

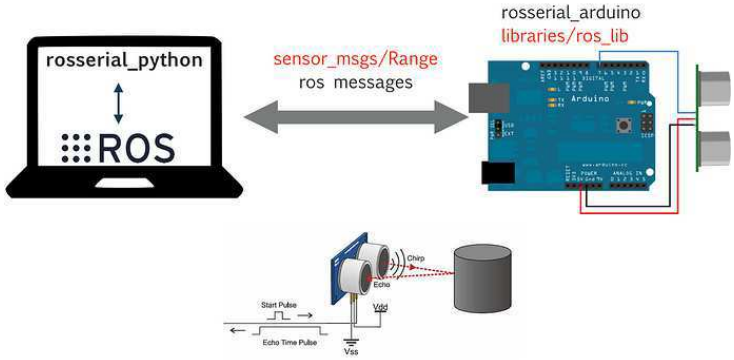

Figura 6 – Ilustração da comunicação Rosserial Arduino

Fonte: (KOUBAA, 2018)

Contudo, neste trabalho a conexão não será através de cabo USB, como de costume, e sim através de uma rede local sobre os protocolos TCPROS com a publicação e subscrição realizada pelos nós criados pelo Rosserial no NodeMcu.

2.2 Plataforma Robótica Falcon

A plataforma robótica da Falcon consiste em um chassi de robô diferencial, contendo duas rodas independentes e um ponto de apoio. As rodas são movimentadas por dois motores 2WD, além disso, alimentados por pilhas alocadas no interior da plataforma, conforme mostrado na **Figura 7**(ROBOCORE, 2018).

Figura 7 – Chassi falco.

Fonte:(ROBOCORE, 2018).

2.3 Hardware

Nesta seção será feita uma descrição sucinta dos componentes de *hardware* utilizados para a montagem do robô.

2.3.1 Host do ROS

O sistema *host* do ROS é o Ubuntu 16.04, neste caso, estamos executando o ROS em notebook Gateway modelo NE56R13b, com as seguintes configurações:

- processador Intel Core i5-3230M;
- monitor de 15,6 HD polegados led/lcd;
- placa gráfica Intel HD Graphics 4000;
- 4 GB DDR3 de Ram;
- SSD 120 GB;
- Placa Wifi 802.11b/g/n.

2.3.2 Arduino

O Arduino é um microcontrolador de *hardware* livre, que facilita na prototipagem de projetos que envolvem circuitos eletrônicos, tem como principais características a sua simplicidade e seu baixo custo, muito usado no meio acadêmico no ensino de programação (THOMSEN, 2014).

2.3.2.1 Arduino Uno

O Arduino Uno, de acordo com (ROBOCORE, 2018), é uma placa microcontroladora baseada no microcontrolador Atmega328, que possui as seguintes características :

- Possui alimentação através do USB, ou externa entre 7 v e 12 v para evitar instabilidade ou sobrecarga;
- Possui conectores de alimentação que fornecem tensão entre 3,3V e 5 v para dispositivos externos e *Shields*;
- O microcontrolador ATMEGA328 , dispositivo de 8bits, arquitetura RISC, 32 de Flash, 2KB de RAM e 1KB EEPROM;
- PWM : 3,5,6,9,10,11 usados com a função analogWrite().

Figura 8 – Arduino Uno

Fonte:(ROBOCORE, 2018)

2.3.2.2 NodeMCU ESP8266-12 V2

O NodeMCU ESP -12 V2 é uma placa com ESP8266 Wifi com uma USB-serial, tal dispositivo é muito utilizado em aplicações de Internet das coisas (IOT) , visto o seu baixo custo e a possibilidade de conectar em rede múltiplos dispositivos a internet(HIRT, 2018).

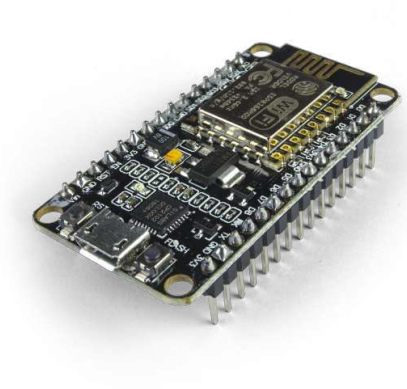

Figura 9 – NodeMCU ESP8266-12 V

Fonte:(HIRT, 2018)

2.3.2.3 Arduino Shield - Motor Driver 2x2A

O *shield motor drive*r permite controlar dois motores DC de 7,5 a 20 V e corrente de até 2A. Essa placa é encaixada sobre o Arduino Uno, possibilitando a conexão de mais dispositivos ao projeto(ROBOCORE, 2015).

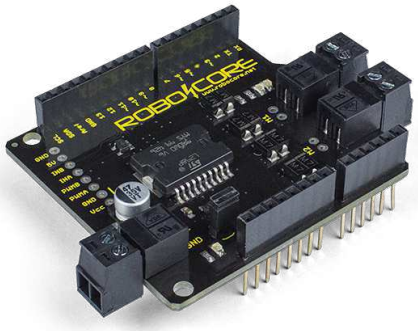

Figura 10 – Arduino Shield - Motor Driver 2x2A

Fonte:(ROBOCORE, 2015)

2.3.2.4 Conversor de Nível Lógico

O conversor lógico é utilizado para conectar dispositivos de diferentes tensões, também é útil na comunicação serial , I2C, SPI (ROBOCORE, 2018).

Figura 11 – Conversor de Nível Lógico

Fonte:(ROBOCORE, 2018)

2.3.2.5 Sensor Ultrasônico

O sensor ultrassônico utilizado é o HS-SR04 que permite leituras de distâncias entre 2cm a 4 metros com precisão de 3mm, sua tensão nos pinos é de 5v (THOMSEN, 2011).

Figura 12 – Sensor Ultrasonico hc-sr04

Fonte: (THOMSEN, 2011)

3 TRABALHOS RELACIONADOS

As aplicações de (MACHADO, 2013) e (PAULA *et al.*,) utilizam minicomputadores como Raspberry Pi com o ubuntu instalado e uma versão do ROS executando nesses terminais: na primeira aplicação, temos as publicações realizadas para o ROS por um *smartphone*; já na aplicação de (PAULA *et al.*,), temos o NodeMcu transmitindo os dados através do protocolo mqtt para um nó interpretador que realiza a publicação e subscrição dos dados coforme a **Figura 13**.

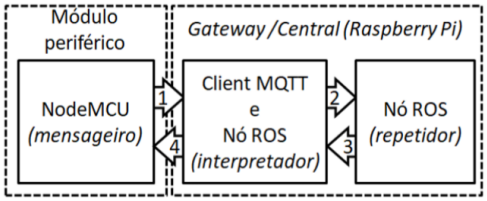

Figura 13 – Diagrama de comunicação dos dados.

Fonte: (PAULA *et al.*,)

A vantagem de se usar minicomputadores é a capacidade de processamento embarcada em um robô aliado ao baixo consumo de energia, entretanto, estes projetos se diferem deste trabalho em relação ao uso de minicomputadores e pelo fato do NodeMcu não realizar publicações diretamente aos nós.

Já (NUNEZ, 2017), utilizou o pacote Rosserial Arduino implementando um nó no NodeMcu-12V que realiza publicações e subscrição de dados para si mesmo usando os serviços básicos do Master que pode estar sendo executado tanto em um mini-computador, como em um notebook ou um desktop .

A implementação de (NUNEZ, 2017) se assemelha ao deste trabalho nas publicações realizadas para o *Master* pelo NodeMcu, a diferença está em haver um tratamento da informação publicada pelo NodeMcu no nó Controle que fica responsável por enviar uma resposta a cada solicitação do robô.

4 DESENVOLVIMENTO

Nesta seção, teremos como tema:

- Implementação;
- Montagem mecânica do robô;
- Montagem eletrônica;

4.1 Implementação

A realização da implementação se dará com a instalação do ROS no Ubuntu, e posteriormente com a criação do espaço de trabalho e dos pacotes, também deverá ser adicionada ao NodeMcu a biblioteca ros_lib proveniente do Pacote Rosserial Arduino.

Realizados estes procedimentos, o desenvolvedor poderá iniciar o desenvolvimento dos softwares do robô em diferentes plataformas Rost do ROS, NodeMcu e Arduino Uno.

Nesta seção, será descrita a função de cada algoritimo implementado nas plataformas, os códigos estarão disponíveis no apêndice A, já no apêndice B, estarão os links dos tutoriais que auxiliaram na instalação e configuração ROS e no desenvolvimento dos softwares do robô.

4.1.1 Implementação do Nó Controle no rost do ROS

Com a criação do espaço de trabalho e posteriormente dos pacotes, conforme descrito no **Apêndice B**, foi realizada a implementação do nó Controle em C++, cuja função é: dar uma nova rota para o robô com a detecção de uma colisão.

A **Figura 14**, demonstra como é realizada a troca de mensagens entre o NodeMcu e o nó Controle, os nós se registram no *master* e caso tenha uma colisão o NodeMcu realiza uma publicação com a palavra colisão, a mensagem é sobrescrita pelo nó Controle, que publica uma nova rota.

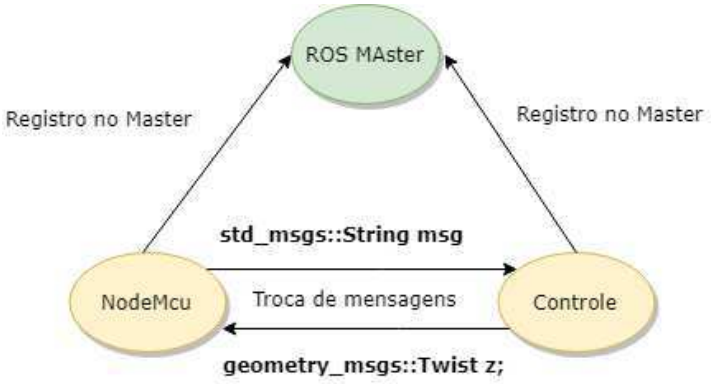

Figura 14 – Inlustração de troca de mensagens.

Fonte : (Autor, 2019).

4.1.2 Implementação no NodeMcu

A inclusão da biblioteca ros_lib possibilitou implementar no NodeMcu dois objetos: o de publicação, que publica a informação de colisão após ser detectada pela função de distância e o objeto de subscrição, que recebe a nova rota públicada pelo nó Controle, e em seguida é transmitida ao Arduino Uno através do protocolo SPI (Serial Peripheral Interface).

A nova rota é determinada pelo nó controle através da função sentido, que realiza o sorteio de um número e verifica se é par ou ímpar. Caso seja par vira à direita, senão, virar a esquerda.

4.1.3 Implemetação do software de comando de motores no Arduino Uno

O Arduino controla a placa Motor Driver através da Biblioteca FalconRobo, não havendo colisão detectada pela sensor ultrassônico no Arduino, é determinado que o robô ande em linha reta, caso contrário, o Arduino determina que o robô pare há espera de uma nova direção.

Com o novo sentido publicado pelo nó Controle, este é subscrito no NodeMcu e transmitida pelo Protocolo SPI ao Arduino que através da placa Motor Driver direciona o robô para um novo caminho.

4.2 Montagem mecânica do Robô

A **Figura 15a** mostra a relação de todas as partes necessárias para montagem do chassi, a **Figura 15b** exibe o resultado final da montagem, para mais detalhes consultar o apêndice B.

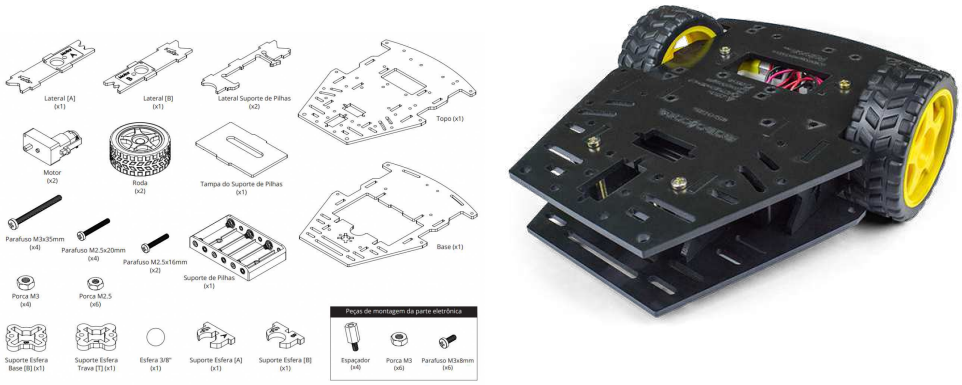

Figura 15 – Relação de peças 15a. Chassi falco 15b.

Fonte: Kit Iniciante p/ Robótica

4.3 Montagem eletrônica

4.3.1 Conexão entre o Arduino e o NOdeMcu através do protocolo Serial Peripheral Interface SPI

A **Figura 16** ilustra as conexões do protocolo SPI, utilizado para a conexão entre os microcontroladores do Arduíno e o NodeMcu. O SPI geralmente é utilizado para a conexão entre um dispositivo mestre e escravo, onde o mestre comanda a comunicação. Além disso, permite a conexão entre múltiplos escravos e um mestre.(ELECTRONICWINGS, 2018).

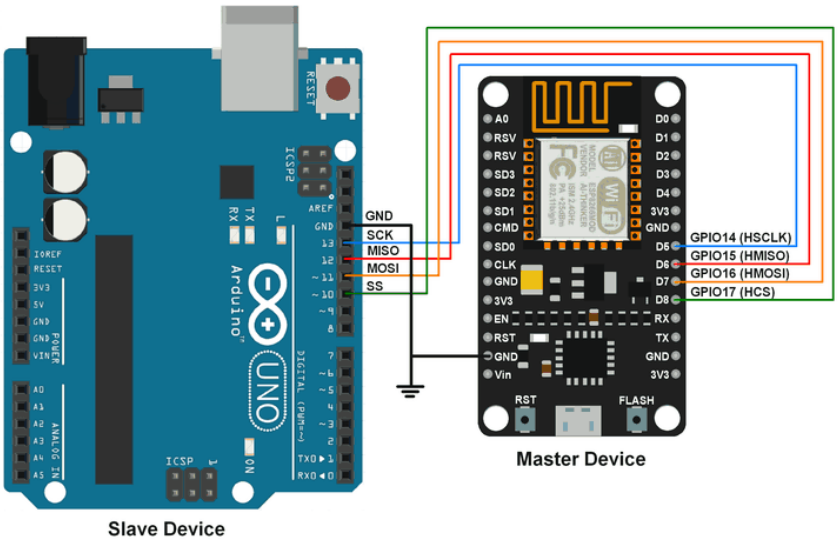

Figura 16 – Conexões do protocolo SPI

Fonte: (ELECTRONICWINGS, 2018).

4.3.2 Conexões entre NodeMcu, sensor Ultrassônico e o Arduino Uno.

A **Figura17** , conforme (KENSHIMA, 2017), ilustra a conexão entre as diferentes placas do projeto: os fios vermelhos e pretos são de alimentação e os fios verdes e amarelos são do sensor ultrassônico, sendo respectivamente conexões do trig e Echo das funções responsáveis pelos cálculos de distância nas placas do Arduino Uno e NodeMcu.

O *trig* ou *trigger* faz a emissão das ondas ultrassônicas, o *echo* recebe esta ondas ultrassônicas, o Arduino Uno e o NodeMcu medem o tempo de envio e recebimento das ondas ultrassônicas, assim, obtém a distância que o obstáculo está(THOMSEN, 2011).

O conversor de tensão foi empregado para permitir que o NodeMcu que opera com 3.3V de tensão, pudesse receber o trigger e o echo do sensor ultrassônico que possui uma tensão de 5V, desse modo, a tensão de 5V do sensor ultrassônico é convertida em 3.3V para o NOdeMcu.

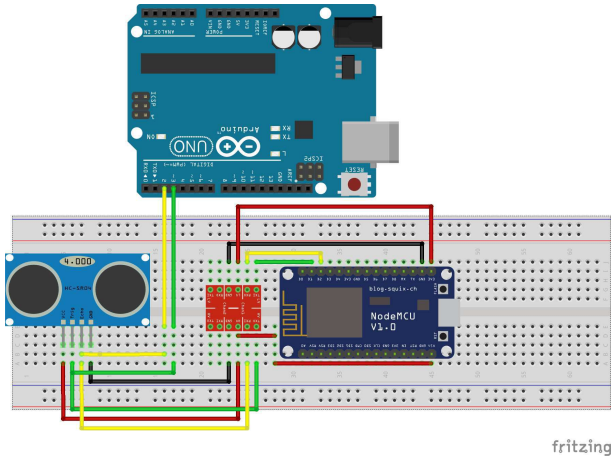

Figura 17 – Conexão entre as placas do robô.

Fonte: (Autor, 2019)

4.3.3 Conexão entre o Arduino Uno e o Shield Motor

A **Figura 18** expõe a conexão entre o Arduino Uno e o Arduino Shield - Motor Driver 2x2A.

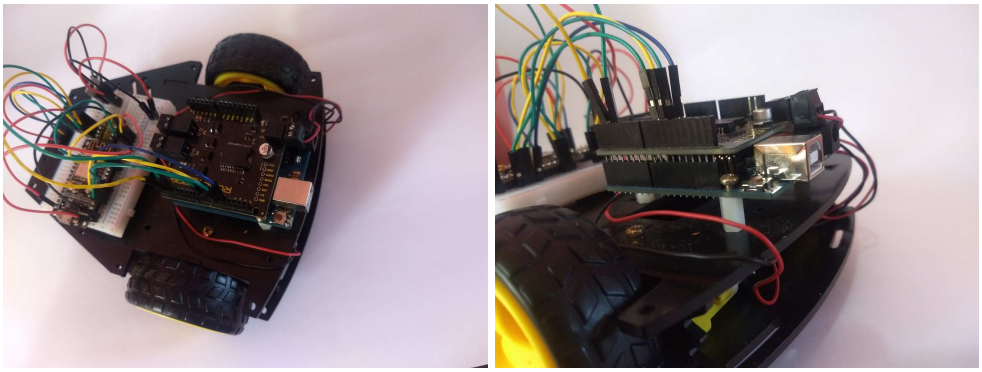

Figura 18 – Arduino Shield - Motor Driver 2x2A sobre o Arduino Uno

Fonte: (Autor,2019).

4.3.4 Fontes de alimentação do Robô

As fontes de energia do robô são: uma bateria de 9V e 6 pilhas AA, conforme **Figura 19**.

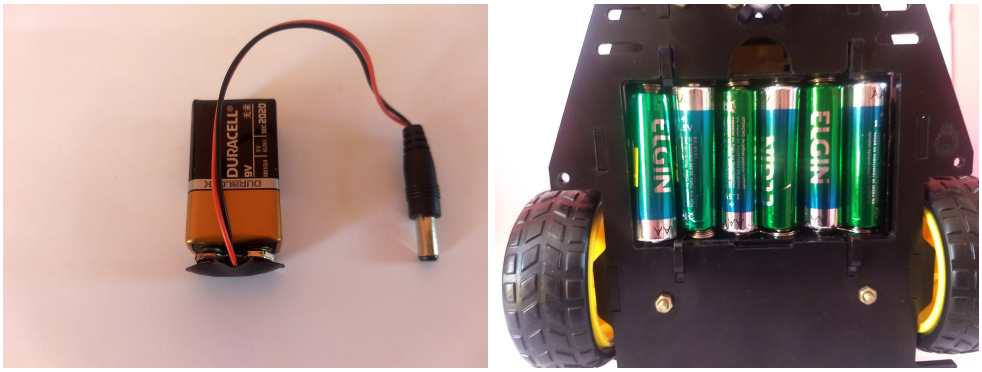

Figura 19 – Fontes de energia do Robô

Fonte: (Autor,2019).

4.3.4.1 Imagem de conexão da bateria de 9V.

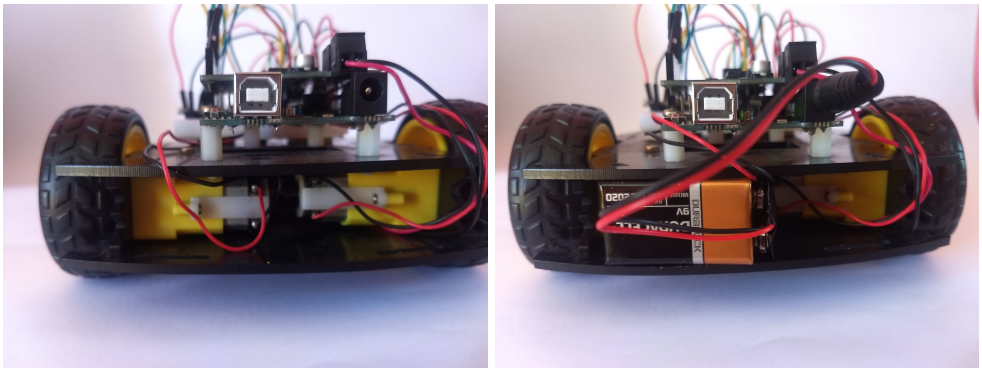

Figura 20 – Conexão da bateria.

Fonte: (Autor, 2019).

- Conexão do NodeMcu à fonte de alimentação.

Na **Figura 21**, temos a conexão do NodeMcu com os pinos de de 5v e o GND, através de um conector USB mini B.

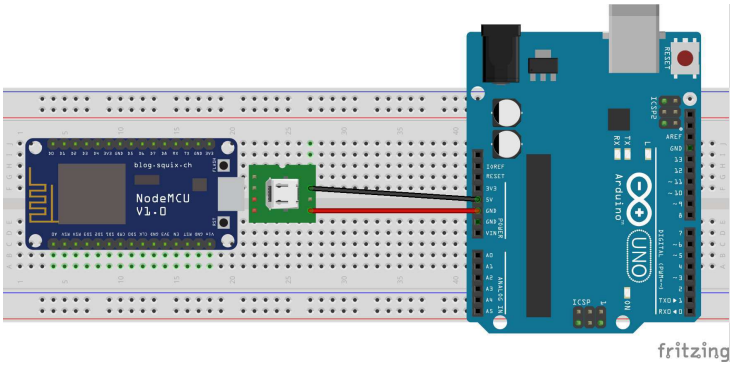

Figura 21 – Conexão do NodeMcu a fonte de alimentação.

Fonte: (Autor, 2019).

5 RESULTADOS E DISCUSSÃO

Neste capítulo, iremos apresentar os resultados obtidos em cada uma das etapas que compõem o desenvolvimento do trabalho, serão relatadas algumas peculiaridades da implementação.

5.1 Rost do ROS

A implementação do nó Controle não trouxe problemas para o projeto, o código foi adaptado ás necessidades, a partir dos tutoriais do *Chatter* e *Listener*(JUNIOR, 2016). No entanto, o usuário do framework deve estar com os conceitos de publicação e subscrição bem compreendidos.

Os testes de conexão do nó controle com o NodeMcu foram realizados com o desenvolvimento de *Chatter* no Nodemcu, que realizou publicações para o nó Controle.

5.2 NodeMcu

O desenvolvimento com o NodeMcu é simples: materiais de instrução, de instalação e funcionamento são bastante difundidos em sites especializados, além de exemplos na própria documentação da placa e guias de usuários.

Os testes foram realizados a partir de publicações realizadas pelo NodeMcu, o qual, recebia subscrições do nó controle e ligava um led, assim foi possível verificar a chegada das respostas do nó Controle, e posteriormente os comandos para os motores.

5.3 Arduino Uno

O arduino uno é responsável pelo controle dos motores, o código necessitou de pequenas mudanças para alterar a direção, pois havia dificuldades de parar diante de um objeto.

Assim, foi criada uma função de detecção de colisão no Arduino Uno, que para o robô diante de uma colisão, o mantém parado aguardando uma resposta do nó Controle e comede a aceleração através de um loop que inicia em 40 e termina em 50, sendo que, a cada incremento de velocidade é realizada uma verificação de colisão.

5.4 Conexões entre as placas

Dada as diferenças de tensões, o uso do conversor de tensão foi uma solução simples e fácil. Porém, o uso do protocolo SPI trouxe dúvidas, apesar da diferença de tensão entre o Arduino Uno e NodeMcu, 5 e 3.3V, o NodeMcu não foi danificado, foram realizados vários testes, e verificou-se a documentação da placa e de tutoriais, ficando claro que em transmissão de dados o NodeMcu tem tolerância a 5v.

5.5 Resultado final

O resultado final foi um robô que tinha uma nova rota pública pelo nó Controle a cada obstáculo detectado, nas **Figuras 22, 23, 24** é exposto o resultado das montagens eletrônica e mecânica.

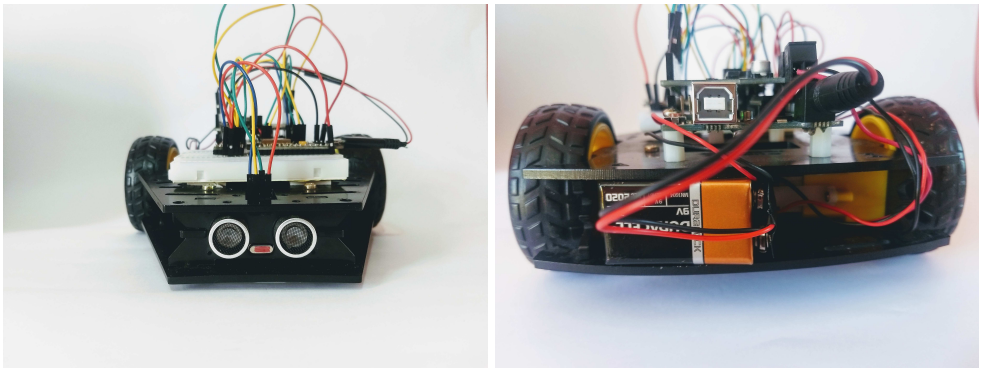

Figura 22 – Parte da frente e de trás do robô.

Fonte: (Autor, 2019).

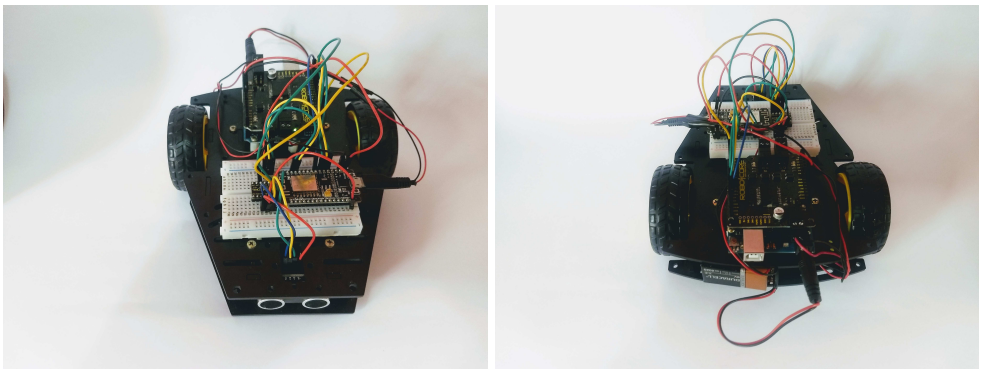

Figura 23 – Parte superior.

Fonte: (Autor, 2019).

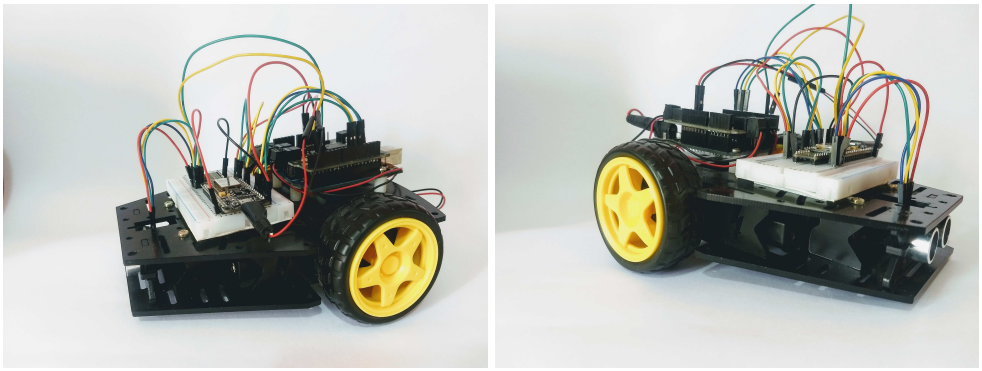

Figura 24 – Laterais do robô.

Fonte:(Autor, 2019).

6 CONCLUSÃO

Dentro dos objetivos propostos que é o emprego do ROS no NodeMcu para o controle de uma plataforma robótica, o ROS se mostrou muito eficiente, pois, facilitou a comunicação entre o computador e o NodeMcu, possibilitando a criação de um robô que interagisse com o computador através da rede Wi-fi.

O ROS é uma ferramenta bastante explorada na comunidade de robótica, ele permite a integração de diversos sensores e atuadores e com a crescente complexidade dos projetos do arduino, este ganha maior processamento atuando como escravo, isto é, informações são publicadas para um computador central, processadas e os comandos são enviados para o arduino e posteriormente para os atuadores.

Porém, o ROS necessita de atualizações para melhorar sua usabilidade no desenvolvimento das aplicações, pois, o uso de terminal para o desenvolvimento da aplicação pode trazer dificuldades, além de aumentar a curva de aprendizagem do usuário.

Contudo, o ROS ainda é uma boa alternativa para quem precisa de maior processamento em seu projeto, com melhorias, que tragam uma interação mais fácil com o NodeMcu poderá se tornar uma ferramenta indispensável para projetos que envolvam o arduino.

REFERÊNCIAS

CERIANI, S.; MIGLIAVACCA, M. **Middleware in robotics**. [S.l.], 2012.

DIRKTHOMAS. **msg**. 2017. Disponível em: ⟨http://wiki.ros.org/msg⟩.

ELECTRONICWINGS. **NodeMCU SPI with Arduino IDE**. 2018. Disponível em: ⟨http://www.electronicwings.com/nodemcu/nodemcu-spi-with-arduino-ide⟩.

HENDRIX, A. **Rosserial arduino/tutoriais**. 2014. Disponível em: ⟨http://wiki.ros.org/rosserial_arduino/Tutorials⟩.

HIRT, C. **Guia gratuito mostra tudo sobre IoT para iniciantes em eletrônica**. 2018. Disponível em: ⟨https://www.filipeflop.com/blog/guia-iot-para-iniciantes-em-eletronica/⟩.

INSTITUTE, S. R. **ROS-Industrial Basic Developer's Training Class**. 2016. Disponível em: ⟨http://ros-industrial.github.io/industrial_training/_downloads/ROS-I%20Basic%20Developers%20Training%20-%20Session%204.pdf⟩.

JUNIOR, F. E. F. **Uma introdução ao Robot Operating System (ROS)**. 2016. Disponível em: ⟨https://www.embarcados.com.br/uma-introducao-ao-robot-operating-system-ros/⟩.

KENSHIMA, G. **NodeMCU ESP8266 com Sensor Ultrassônico HC-SR04**. 2017. Disponível em: ⟨http://blog.baudaeletronica.com.br/integrar-sensor-ao-nodemcu-esp8266/⟩.

KENTAROWADA. **Rostopic**. Disponível em: ⟨http://wiki.ros.org/rostopic⟩.

KOUBAA, A. **Learn rosserial with Arduino with video illustrations**. 2018. Disponível em: ⟨https://discourse.ros.org/t/learn-rosserial-with-arduino-with-video-illustrations/5937⟩.

LAGES, W. F. Robot operating system (ros). 2016.

MACHADO, G. Z. **Desenvolvimento de um kit didático para estudo de robótica: veículo remotamente controlado através da plataforma ROS**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2013.

MAHTANI, A.; SANCHEZ, L.; FERNANDEZ, E.; MARTINEZ, A. **Effective Robotics Programming with ROS**. [S.l.]: Packt Publishing Ltd, 2016.

MATHWORKS. **Exchange Data with ROS Publishers and Subscribers**. 2018. Disponível em: ⟨https://www.mathworks.com/help/robotics/examples/exchange-data-with-ros-publishers.html⟩.

NUNEZ, A. **espros**. 2017. Disponível em: ⟨https://github.com/agnunez/espros⟩.

OLIVEIRA, S. de. **Internet das Coisas com ESP8266, Arduino e Raspberry Pi**. [S.l.]: Novatec Editora, 2017.

PACKT. **ROS ARCHITECTURE AND CONCEPTS**. 2016. Disponível em: ⟨https://hub.packtpub.com/ros-architecture-and-concepts/⟩.

PAULA, F. O. de; BARBOSA, B. H. G.; OLIVEIRA, S. de; HENRIQUE, G. Utilização do protocolo mqtt e framework ros na instrumentação de tratores agrícolas.

ROBOCORE. **Shield Motor 2x2A**. 2015. Disponível em: ⟨https://www.robocore.net/loja/drivers-de-motores/arduino-shield-motor-driver-2x2a⟩.

ROBOCORE. **Learn rosserial with Arduino with video illustrations**. 2018. Disponível em: ⟨https://www.robocore.net/loja⟩.

ROMAINREIGNIER. **Rossserial**. 2018. Disponível em: ⟨http://wiki.ros.org/rosserial⟩.

SAITO, I. Rosnode. In: . [s.n.], 2013. Disponível em: ⟨http://wiki.ros.org/rosnode⟩.

THOMSEN, A. **Como conectar o Sensor Ultrassônico HC-SR04 ao Arduino**. 2011. Disponível em: ⟨ComoconectaroSensorUltrass\let\begingroup\escapechar\m@ne\let\MT@subst@\OT1/ptm/bx/n/12\def{\@@par}⟩.

THOMSEN, A. **O que é Arduino?** 2014. Disponível em: ⟨https://www.filipeflop.com/blog/o-que-e-arduino/⟩.

UNKNOWNENTITY1. **TCPROS**. 2013. ⟨http://wiki.ros.org/ROS/TCPROS⟩.

APÊNDICE A – CODIGO FONTE

A.1 Controle

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "geometry_msgs/Twist.h"
#include <std_msgs/Float32.h>
#include <time.h>
#include <stdint.h>
#include <iostream>

ros::Publisher commandPub;


int sentido(){

    srand(time(NULL));
    int x = rand()%100;

    if(x % 2==0){

        return 1;
    }

    return 0;
}
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
    int s = sentido();

    ROS_INFO("I heard: [%s]", msg->data.c_str());

    geometry_msgs::Twist z;
    z.angular.z= s;
    commandPub.publish(z);
}

int main(int argc, char **argv){
 ros::init(argc, argv,"Controle");

 ros::NodeHandle n;


```

```
  ros::Subscriber sub = n.subscribe("chatter", 1000, chatterCallback);
  commandPub = n.advertise<geometry_msgs::Twist>("cmd_vel", 10);

  ros::spin();

  return 0;
}
```

A.2 NodeMcu

```
      #include <ros.h>
      #include <geometry_msgs/Twist.h>
      #include <Ultrasonic.h>
      #include <std_msgs/String.h>
      #include <SoftwareSerial.h>
      #include<SPI.h>


      char  buff[]=  "D\n";

      char  buff2[]="E\n";



       #define trigPin D1
       #define echoPin D2
       int   NovaDirecao ;
      ////////////////////
      // WiFi Definitions //
      ////////////////////
      // WiFi configuration. Replace '***' with your data
      const char* ssid = " ";
      const char* password = " ";
      IPAddress server(000, 000,00,00);
      const uint16_t serverPort = 11411;

      /////////// Função de  distancia   com base no sensor/////////
      boolean distancia(){

      boolean colicao ;
      long duracao;
      int distancia, aux;
      digitalWrite(trigPin,HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin, LOW);
```

```
36
37        duracao = pulseIn(echoPin, HIGH ,1000000);
38        aux = duracao;
39        distancia = (aux*340/10000)/2;
40        Serial.print("Distancia: ");
41        Serial.println(distancia);
42
43        if(distancia < 15){
44
45          colicao = true;
46
47
48          }
49
50         else if(distancia > 15){
51
52             colicao = false ;
53
54             }
55
56        return colicao;
57
58        }
59
60
61        void setupWiFi(){
62
63        Serial.println("////////////////////////////////////////////////////////");
64
65        // connect to ROS server as as a client
66        Serial.begin(115200);               // Use ESP8266 serial only for to monitor the process
67        Serial.println();
68        Serial.print("Connecting to ");
69        Serial.println(ssid);
70        WiFi.begin(ssid, password);
71        while (WiFi.status() != WL_CONNECTED) {
72        delay(500);
73        Serial.print(".");
74
75         // rst
76        }
77        Serial.println("");
78        Serial.println("WiFi connected");
79        Serial.println("IP address: ");
80        Serial.println(WiFi.localIP());
81
82        }
```

```
83
84        ros::NodeHandle  nh;
85        std_msgs::String dist;
86        ros::Publisher chatter("chatter", &dist);
87
88        //*****Função  direcional do robô****
89
90        void messageCb( const geometry_msgs::Twist& msg){
91          int i;
92
93
94           if(msg.angular.z>0) // virar a   direita
95           {
96
97
98            for(int i=0; i<sizeof buff; i++)
99            SPI.transfer(buff[i]);
100
101
102 //
103            Serial.println("msg = 1 ");
104 //
105
106           }
107           else if(msg.angular.z == 0) // virar a esquerda
108           {
109
110            Serial.println("msg = 0 ");
111
112            for(int i=0; i<sizeof buff2; i++)
113             SPI.transfer(buff2[i]);
114
115
116           }
117
118       }
119       ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel", &messageCb );
120
121       void setup() {
122
123         // put your setup code here, to run once:
124         Serial.begin(115200);
125         SPI.begin();
126         setupWiFi();
127         delay(2000);
128
129
```

```
130         //Configuração  dos pinos do   sensor ultrasonico
131
132         pinMode(trigPin, OUTPUT);
133         pinMode(echoPin, INPUT);
134         digitalWrite(trigPin, LOW);
135
136         // Configuração  do TCP dos ros
137
138         nh.getHardware()->setConnection(server, serverPort);
139
140         //inicializando   o nó  do ros
141
142         nh.initNode();
143         nh.subscribe(sub);
144         nh.advertise(chatter);
145         chatter.publish( &dist );
146
147
148
149       }
150
151       void loop() {
152
153         boolean colisao ;// variavel auxiliar
154         colisao = distancia();
155
156         if(colisao){ // Caso haja   colisao, o nodemcu  realiza  contato  com o nó
157         // para saber  qual vai ser a nova direção
158          //NodeMCU.print(4); // para  o robo  ate  saber a nova  direção
159         char hello[15] = "Colisao";
160         setupWiFi();
161         dist.data = hello;
162         chatter.publish( &dist );
163
164         nh.spinOnce();
165
166         return; // Serve para retorna  ao inicio  da função
167         } // ....
168
169
170         }//  Fim da função void loop()
```

A.3 Arduino Uno

```
#include <Ultrasonic.h>
#include <SPI.h>
#include <FalconRobot.h>

////// VARIAVEIS DE SENTIDO ////

char esquerda = 'E';
char direita = 'D';


//// Cria  o objetos motor nos acionando os pinos  ~5,7 **** ~6,8

FalconRobotMotors motors(5, 7, 6, 8);

//////// variaveis  do SPI

char buff [100];
volatile byte index;
volatile bool receivedone;  /* use reception complete flag */

boolean distancia_movimento(){ //  verifica durante a aceleração se haverá colisão

   Ultrasonic ultrasson(3,2);
   int   distancia = 25; //  distancia minima log
   int  distancia_atual;
   boolean  colisao;
  //  leitura   da distancia  atual
  distancia_atual = ultrasson.Ranging(CM);

  if(distancia_atual <=  distancia){

    return true;
    }


   return false;
    }


    void setup (void)
    {
      Serial.begin (115200);
      SPCR |= bit(SPE);          /* Enable SPI */
      pinMode(MISO, OUTPUT);     /* Make MISO pin as OUTPUT */
```

```
45          index = 0;
46          receivedone = false;
47          SPI.attachInterrupt();    /* Attach SPI interrupt */
48        }
49
50        void loop (void)
51        {
52          if (receivedone && distancia_movimento())          /*verifica  se há colisão e mensagens
53          {
54              buff[index] = 0;
55
56              Serial.println(buff);
57              index = 0;
58              receivedone = false;
59
60
61                 if(buff[0]==direita){
62
63                    //motors.stop();
64                    //delay(250);
65
66                    motors.drive(50,BACKWARD);
67                    delay(250);
68                    motors.stop();
69                    delay(250);
70                    motors.leftDrive(50, FORWARD);
71                    motors.rightDrive(50, BACKWARD);
72                    delay(250);
73                    motors.stop();
74                    delay(250);
75
76
77                  }// direita
78
79                  else if(buff[0] == esquerda){
80
81
82                     //motors.stop();
83                     //delay(250);
84                     motors.drive(50,BACKWARD);
85                     delay(250);
86                     motors.stop();
87                     delay(250);
88                     motors.rightDrive(50, FORWARD);
89                     motors.leftDrive(50, BACKWARD);
90                     delay(250);
91                     motors.stop();
```

```
92                    delay(250);
93
94                   } // esquerda
95
96            }
97            else { // caso não haja colisão
98
99           motors.stop();
100         delay(250);
101         int acelerar = 0;
102
103         for(acelerar= 40 ;acelerar <  51;acelerar ++){
104
105          if(distancia_movimento()){
106
107           Serial.println("colisao no movimento");
108
109          motors.stop();
110          motors.drive(0, FORWARD);
111          return; //  voltar  no inicio   do loop ;
112
113           } // if
114         motors.drive(acelerar, FORWARD);
115
116         delay(100);} /// fim  do loop
117           motors.stop();
118         delay(250);
119
120         } // fim do if receivedone && distanciaatual
121
122
123       }// fim do void loop()
124
125       // SPI interrupt routine
126       ISR (SPI_STC_vect)
127       {
128         uint8_t oldsrg = SREG;
129         cli();
130         char c = SPDR;
131         if (index <sizeof buff)
132         {
133           buff [index++] = c;
134           if (c == '\n'){     /* Check for newline character as end of msg */
135            receivedone = true;
136           }
137         }
138         SREG = oldsrg;
```

```
139         }// http://www.electronicwings.com/nodemcu/nodemcu-spi-with-arduino-ide
```

APÊNDICE B – LINKS DOS TUTORIAIS

Durante o desenvolvimento pratico do projeto encontramos diversos tutoriais, que nos ajudaram no desenvolvimento do robô. Os explicativos podem ser consultados nos links dispostos no apêndice D, sendo eles:

B.1 Instalação do ROS

Neste trabalho foi utulizada a Kinectic mas o ROS possue outras versões.

- Página oficial do ROS : http://wiki.ros.org/kinetic/Installation .
- Site Embarcados: https://www.embarcados.com.br/uma-introducao-ao-robot-operating-system-ro/ .

B.2 Istalacão do Rosserial Arduino

- Tutoriais Rosserial Arduino http://wiki.ros.org/rosserial_arduino/Tutorials
- Site Embarcados: https://www.embarcados.com.br/arduino-com-rosserial_arduino/
- O livro : Effective Robotics Programming with ROS Third Edition

B.3 Implementacãoo do nó Controle

- Tutoriais sobre o ROS: http://wiki.ros.org/pt_BR/ROS/Tutorials
- ROS.org : http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber
- Site Embarcados: https://www.embarcados.com.br/criando-um-hello-world-no-robot-operating-system/ .

B.4 Implementacão do Rosserial Arduino no NodeMcu

- GitHub de Agnunes: https://github.com/agnunez/espros
- O livro : Effective Robotics Programming with ROS Third Edition

B.5 Implementacão SPI

- Electronicwings : http://www.electronicwings.com/nodemcu/nodemcu-spi-with-arduino-ide
- Página Hélio Souza Mendoça : https://paginas.fe.up.pt/ hsm/docencia/comp/spi-e-i2c/

B.6 Implementacão do algorimo de controle de motores

- Site Robocore: https://www.robocore.net/tutorials/kit-iniciante-robotica-primeiros-movimentos.html .

B.7 Conexões entre NodeMcu, sensor Ultrassônico e o Arduino Uno.

- Blog Baú da eletrônica: http://blog.baudaeletronica.com.br/integrar-sensor-ao-nodemcu-esp8266/ .

APÊNDICE C – COLOCANDO O ROBÔ PARA FUNCIONAR

- Colocar as 6 pilhas e conectar a bateria no robô e colocá-lo de frente a um obstáculo.
- Executar o comando descritos abaixo em terminais distintos.
- Arquadar até que NodeMcu se conecte ao servidor do ROS.

C.0.1 O Comando cd /catkin_ws/catkin_make

Serve para compilar os arquivos do pacote.

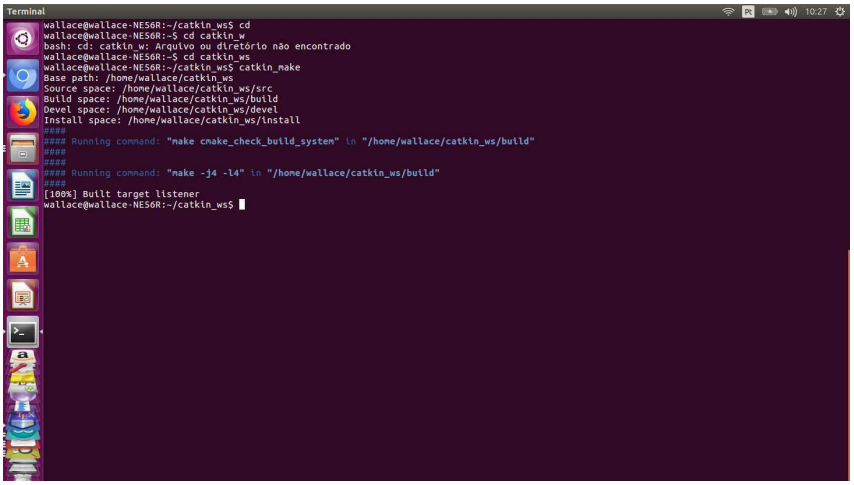

Figura 25 – comando catkin_make

Fonte:(autor,2019).

C.0.2 O comando source /catkin_ws/devel/setup.bash

Iniciar os nós do ROS.

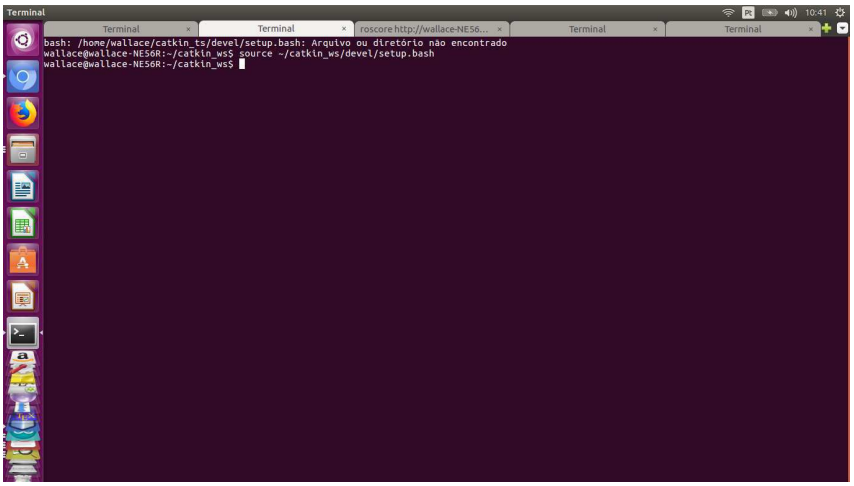

Figura 26 – Comando source /catkin_ws/devel/setup.bash

Fonte: (autor,2019).

C.0.3 O comando roscore

Serve para inicializar o ROS .

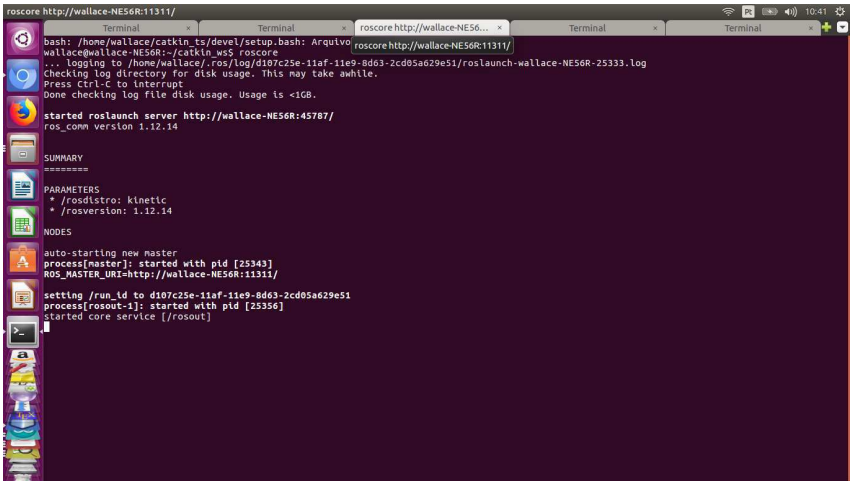

Figura 27 – comando roscore

Fonte: (autor,2019).

C.0.4 O comando cd /catkin_ws source ./devel/setup.bash rosrun tarefa1 Controle

Neste caso, o usuário executará o nó Controle no pacote tarefa1 .

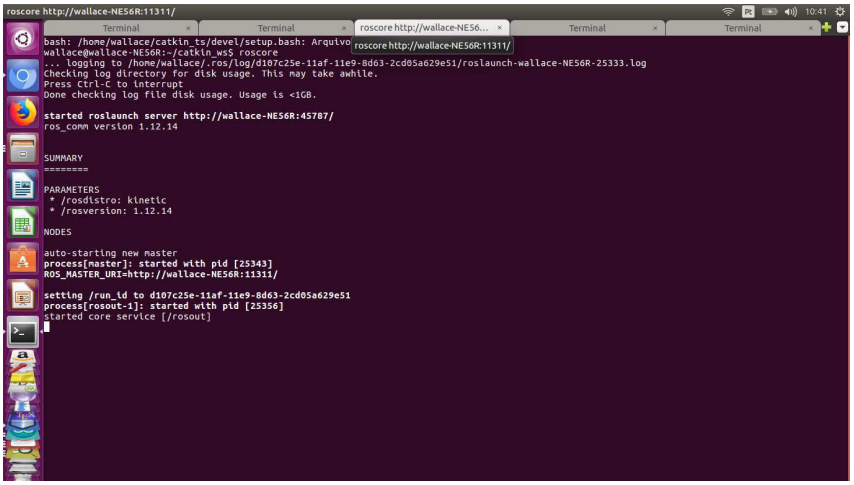

Figura 28 – Comando source ./devel/setup.bash

Fonte: (autor,2019).

O **rosrun rosserial_python serial_node.py tcp**, comando que realiza a conexão entre o robô e o Master.

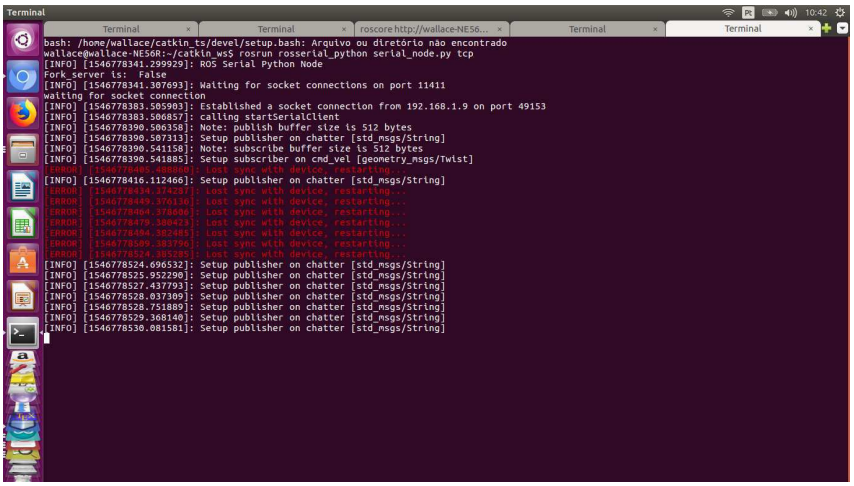

Figura 29 – Comando que executa o Rosserial

Fonte: (autor,2019).